Design und Entwicklung eines Multimedia Home-Entertainment Systems

Marc Klein

Diplomarbeit

nach einem Thema von
Prof. Dr.-Ing. Philipp Slusallek
Naturwissenschaftlich-Technische Fakultät I
Fachrichtung 6.2 – Informatik
Universität des Saarlandes, Saarbrücken, 2003

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Saarbrücken, 31. Januar 2003

Ich danke...

Prof. Slusallek für die Vergabe des Themas der Diplomarbeit.
Marco Lohse für die ausgezeichnete Betreuung.
Patrick Becker, Patrick Cernko, Wolfgang Enderlein und Markus Sand, die Vorarbeiten zu dieser Arbeit geleistet haben.
Meinen Eltern, die mir dieses Studium ermöglichten.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Ziele . . . 1
- 1.2 Inhalt . . . 4
- 1.3 Hinweis . . . 5

2 Verwandte Projekte und Systeme **6**
- 2.1 Open Source . . . 6
  - 2.1.1 MPlayer . . . 6
  - 2.1.2 Alsaplayer . . . 7
  - 2.1.3 GStreamer Player . . . 8
  - 2.1.4 Video Disk Recorder . . . 8
  - 2.1.5 Vergleichbare Systeme . . . 9
- 2.2 Ansätze aus der Forschung . . . 10
  - 2.2.1 SAMBITS . . . 10
  - 2.2.2 Media/Entertainment Gateway . . . 10
  - 2.2.3 KOM Player . . . 11
  - 2.2.4 Interaktives TV System . . . 12
  - 2.2.5 CustomTV . . . 12
  - 2.2.6 TV Anytime Forum . . . 13
- 2.3 Multimedia Home Plattform . . . 13

3 Die Netzwerk-Integrierte Multimedia Middleware (NMM) **15**
- 3.1 Übersicht . . . 16
- 3.2 Flussgraph . . . 16
  - 3.2.1 Formate . . . 18
  - 3.2.2 Knoteneinteilung . . . 19
- 3.3 Nachrichten . . . 21
- 3.4 Speicherverwaltung . . . 23
- 3.5 Die Entwicklung von NMM Knoten . . . 23
  - 3.5.1 Beispiel . . . 26

4 Auswahl der Hardware **31**
4.1 Anforderungen an die Hardware 31
4.2 Komponenten 32
4.2.1 Hardware im Überblick 32
4.2.2 CPU und Mainboard 33
4.2.3 Netzwerk 34
4.2.4 Grafikkarte 34
4.2.5 Sound 35
4.2.6 DVD Laufwerk 35
4.2.7 Festplatte 35
4.2.8 DVB Karte 36
4.2.9 MPEG2 Enkoderboard 36
4.2.10 Fernsehgerät 37
4.3 Zusätzliche Hardware 37
4.3.1 Infrarotempfänger 38
4.3.2 Liquid Crystal Display (LCD) 39
4.3.3 Gehäuse und Geräuschminimierung 39

5 Software Komponenten **40**
5.1 Grafisches Benutzerinterface 41
5.1.1 Windowsystem 43
5.1.2 Widgets 47
5.1.3 Schnittstellen der Widget-Klasse 48
5.1.4 Composite Widget 52
5.1.5 Vorarbeiten zur Entwicklung der Basis-Widgets 52
5.1.6 Basis-Widgets 56
5.1.7 Decorator 62
5.1.8 Widgetunits 65
5.2 Knoten 69
5.2.1 Quellknoten 69
5.2.2 Verarbeitungsknoten 73
5.2.3 Senkeknoten 93
5.3 Benutzereingaben und Ausgaben 97
5.3.1 Infrarotfernbedienung - LircProducer 97
5.3.2 XProducer 98
5.3.3 Ansteuerung des LC-Displays 99

6 Die Multimedia-Box Anwendung **100**
6.1 Bedienung der Multimedia-Box Anwendung 102
6.1.1 Haupt- und Untermenü 102
6.1.2 DVD-Spieler 103
6.1.3 CD-Spieler 104
6.1.4 CD-Grabber 105
6.1.5 TV-Viewer 106

6.1.6 TV-Timer . . . 107
6.1.7 Playlist . . . 108
6.1.8 MP3-Spieler . . . 109
6.1.9 Taskmanager . . . 110
6.1.10 Konfiguration . . . 111
6.2 Anwendungs-Framework . . . 112
6.2.1 Architektur der Hauptanwendung . . . 112
6.2.2 Anwendungs-Objekt . . . 113
6.2.3 Globale Knoten . . . 115
6.2.4 XML Parser . . . 116
6.2.5 Integration der Zustände in die Anwendung . . . 117
6.2.6 Statewechsel . . . 120
6.3 Implementierung der Zustände . . . 122
6.3.1 AutoMenu . . . 122
6.3.2 DvdState . . . 124
6.3.3 CdState . . . 126
6.3.4 GrabState . . . 126
6.3.5 DvbState . . . 127
6.3.6 MP3State . . . 129
6.3.7 Playlist . . . 130
6.3.8 TV Timer . . . 134
6.3.9 ConfigState . . . 137
6.3.10 TaskMgrState . . . 138
6.3.11 Erweiterungen . . . 140

7 Zusammenfassung und Ausblicke 146
7.1 Erzielte Ergebnisse . . . 146
7.2 Ausblick . . . 148
7.2.1 Neue Oberfläche . . . 148
7.2.2 Neue Funktionen . . . 149
7.2.3 Ressourcen-Management . . . 149
7.2.4 Verteilte Anwendungen . . . 149
7.2.5 Benutzerinteraktion . . . 150

A Tools und Treiber 151
A.1 TV-Out bei NVidia Grafikkarten . . . 151
A.2 Soundblaster Live . . . 151
A.3 DVD Laufwerksoptionen . . . 152
A.4 Installation der DVB Karte . . . 153
A.4.1 Treiber . . . 153
A.4.2 Konfiguration . . . 153
A.5 Infrarotempfänger . . . 154
A.5.1 Schaltplan . . . 154
A.5.2 Treiberinstallation . . . 155

A.6 LC-Display 156
A.6.1 Hardware 156
A.6.2 Treiber 157
A.7 MMBox Konfiguration 158
A.7.1 Voreinstellungen 158
A.7.2 XML Beschreibung der Zustände und Tastenzuordnung . 159
A.8 Tastenbelegung 164
A.9 AC3 Frame Codes 164

Kapitel 1

Einleitung

1.1 Ziele

Das Fernsehgerät wurde in den letzten Jahren nicht wesentlich um neue Funktionen erweitert, jedoch gibt es eine ganze Reihe zusätzlicher Geräte, die sich u.a. an den Fernseher anschließen lassen. Heutzutage besitzt fast jeder einen Videorekorder, um Fernsehsendungen aufnehmen zu können und Aufnahmen beliebig oft abzuspielen. Analoge Geräte werden durch digitale ersetzt, beispielsweise haben CD-Player mittlerweile Schallplatten und Kassetten abgelöst, mit einem CD-Brenner kann man ohne Qualitätsverlust Musikstücke auf eine CD archivieren. Aber auch die analogen Videorekorder werden allmählich durch digitale Geräte ersetzt. DVD Player werden zum Abspielen von Video- und Audiodaten benutzt, aber auch analoge Videoaufnahmen werden durch digitale Videorekorder abgelöst. Diese digitalen Videorekorder speichern die Aufnahmen auf einer Festplatte oder schreiben sie direkt auf eine DVD.

Auch das analoge Fernsehen wird nach und nach durch digitales Fernsehen ersetzt und erhält Einzug in viele Haushalte. Filme können für eine bestimmte Uhrzeit bestellt werden (Video On Demand) oder laufende Sendungen können in verschiedenen Perspektiven betrachtet werden.

Auch Radioprogramme können nicht nur über die normale Antenne empfangen werden. Sie werden teilweise auch digital über Satellit ausgestrahlt und können mit einem geeigneten Receiver empfangen werden.

Durch die ständig wachsende Bandbreite, mit der Heimanwender Zugriff auf Internetdienste haben, können auch Fernseh- und Radioprogramme über eine Internetverbindung übertragen werden.

Alle oben genannten Dienste sind nicht nur Prototypen, sondern werden schon zahlreich eingesetzt. Zum Empfang oder zur Verarbeitung dieser Daten wird ein speziell dafür entwickeltes Gerät benötigt. Es gibt Geräte, die mehrerer Funktionen vereinigen und die unterschiedlichsten Daten empfangen und verarbeiten können, es gibt aber keine Komplettlösung, die alle Funktionen abdeckt und zusätzlich noch erweiterbar ist.

Die Bedienung der Geräte gestaltet sich sehr unterschiedlich. Jedes besitzt eine eigene Fernbedienung und eine eigene Möglichkeit zur Interaktion und Steuerung.

Dienste und Informationen wachsen zusammen und die Darbietung auf einem gemeinsamen Endgerät ist wünschenswert. Ein PC ist die ideale Plattform für diese Medienkonvergenz, entsprechende Hardware und Software vorausgesetzt. Mit einem DVD-ROM Laufwerk hat man die Voraussetzung geschaffen, Video-DVDs oder Audio-CDs abzuspielen. CD- oder DVD-Brenner erlauben die Archivierung großer Datenmengen, um etwa Radio- oder Fernsehsendungen zu archivieren. Eine digitale TV-Erweiterungskarte ermöglicht den Empfang von digitalen Fernsehprogrammen über Satellit oder Kabel. Entsprechende Lösungen gibt es auch für analog empfangene TV Programme. Eine Netzwerkkarte oder Modem erlaubt den Zugriff auf verschiedene Internetdienste.

Der PC ist ein offenes System und lässt sich beliebig erweitern, dadurch sind neue Medienformate zuerst für den PC verfügbar und werden dann erst in ein Endgerät integriert. Die Integration der Funktionen von mehreren Geräten in einen PC führt dazu, dass neue Funktionen entstehen, die vorher nicht möglich waren. Beispielsweise können digitale Satellitenprogramme direkt auf die PC-Festplatte geschrieben werden und dort so umgewandelt werden, damit sie direkt auf eine CD oder DVD geschrieben werden können.

Ein handelsüblicher PC genügt jedoch insbesondere den optischen Anforderungen an ein Endgerät nicht. Einerseits lässt sich der PC auf Grund seines Designs nicht ins Wohnzimmer integrieren, wie z.B. ein Videorekorder. Auch die Geräuschentwicklung eines Standard-PC ist viel zu hoch, um ihn als Multimedia-Gerät ins Wohnzimmer zu stellen. Andererseits gibt es keine Software, die Funktionen wie DVD/CD Abspielen und den Empfang und die Aufnahme von digitalem Fernsehen vereinigt und zusätzlich noch leicht erweiterbar und konfigurierbar ist. Die Interaktion der Software mit dem Benutzer wird meist mit der Maus durchgeführt. Ein Endgerät muss hingegen mit einer Fernbedienung gesteuert werden können und die grafische Benutzeroberfläche an den Fernseher angepasst sein.

Ziel dieser Arbeit ist nun, die Software für ein Multimedia-Endgerät auf Basis eines Standard-PC zu entwickeln. Diese Software trägt die Bezeichnung Multimedia-Box. Sie soll an die Anforderungen eines Endgerätes angepasst sein. Im Rahmen der vorliegenden Arbeit sollen folgende Ziele erreicht werden:

- Auswahl von Standard-PC-Komponenten, mit denen die Multimedia-Box aufgebaut wird. Dabei wird besonderer Wert auf das Design und leise Laufgeräusche gelegt (vgl. Abbildung 1.1).

  Die Multimedia-Box soll mit einer Fernbedienung gesteuert und an einen Fernseher angeschlossen werden können. Der Empfang von TV-Sendungen sollte möglich sein, ebenso das Auslesen einer DVD und CD.

  Es sollten auch Audio-Anschlüsse vorhanden sein, die eine Verbindung mit einer Stereoanlage oder einem Digital-Receiver ermöglichen.

- Entwicklung eines Toolkits zum Erstellen einer grafischen Benutzerschnitt-

Abbildung 1.1: Die Multimedia-Box wird an einen normalen Fernseher angeschlossen und lässt sich komplett über eine Fernbedienung steuern.

stelle. Dieses Toolkit soll insbesondere die Anforderungen von Home-Entertainment Szenarien berücksichtigen. Es soll hierarchische Menüs zur Verfügung stellen und Textausgaben mit verschiedenen Zeichensätzen unterstützen (vgl. Abbildung 1.2). Listenelemente sollen mehrere Textzeilen darstellen können, wobei einzelne Zeilen hervorgehoben werden können. Alle grafischen Elemente müssen in ihrer Größe veränderbar sein.

- Folgende Funktionen soll die Multimedia-Box bieten:
  - Abspielen von DVDs, wobei die auf den Video-DVDs vorhandenen grafischen Menüs angezeigt und interpretiert werden sollen.
  - Empfang, Aufnahme und Wiedergabe von TV-Sendungen, wobei auch das Pausieren, Vor- und Zurückspulen (Timeshifting) unterstützt werden soll.
  - Abspielen von Audio-CDs und deren Archivierung auf Festplatte.
  - Integration einer Playlist, mit der verschiedene Dateien in unterschiedlichen Formaten abgespielt werden können.
  - Taskmanager, der das Hin- und herschalten zwischen gleichzeitig ausgeführten Funktionen erlaubt und Funktionen beenden kann.
- Entwicklung eines Anwendungs-Framework, das eine modulare Architektur besitzt, so dass die Anwendung leicht erweitert werden kann. Funktionen

Abbildung 1.2: Zwei ganz unterschiedliche Präsentationen des Multimedia-Box Hauptmenüs. Die Gestalt (Skins) der Menüstruktur lässt sich individuell anpassen.

sollen leicht in die Anwendung integriert werden können. Grafische Elemente und das Verhalten der Funktionen sollen sich über eine einheitliche Schnittstelle konfigurieren lassen.

Das Framework soll auch die Ausführung mehrere Funktionen gleichzeitig erlauben, dabei müssen Mechanismen entwickelt werden, die beispielsweise verhindern, dass zwei Funktionen das gleiche Geräte beanspruchen, das nur einmal zur Verfügung steht, wie z.B. eine Grafikkarte.

Benutzereingaben sollen über verschieden Geräte möglich sein, wie z.B. Tastatur oder Fernbedienung und erweiterbar sein, so dass auch andere Eingabemöglichkeiten, wie beispielsweise Spracherkennung verwendet werden können.

1.2 Inhalt

Kapitel 2 stellt einige mit dieser Arbeit verwandten Systeme vor. Abschnitt 2.1 beschäftigt sich mit einigen Open Source Projekten, die Audio- oder Videodaten verarbeiten können. Sie werden mit den Zielen und Ansätzen der Multimedia-Box verglichen. In Abschnitt 2.2 werden diese mit einigen Forschungsprojekten verglichen. Die Multimedia Home Plattform, die eine Schnittstelle für die Entwicklung von interaktiven digitale TV-Programme bietet, wird in Abschnitt 2.3 erläutert.

In Kapitel 3 wird die Netzwerk-Integrierte Multimedia Middleware (NMM) vorgestellt, die die Grundlage für die Entwicklung der Multimedia-Box Anwendungen bildet. Abschnitt 3.1 gibt einen Überblick über die Bestandteile von NMM. In Abschnitt 3.2 wird die Funktionsweise einer NMM-Anwendung erläutert, insbesondere der Datenfluss einer Anwendung. Abschnitt 3.3 beschäftigt sich mit dem NMM-Nachrichtensystem und Abschnitt 3.4 mit der Speicherverwaltung. In Abschnitt 3.5 wird die Entwicklung eines NMM-Knotens beschrieben und an Hand eines Beispiels gezeigt, wie eine NMM-Anwendung geschrieben wird.

Kapitel 4 beschäftigt sich mit der Auswahl der PC-Komponenten für die Multimedia-Box. Abschnitt 4.1 beschreibt alle verwendeten Hardware-Teil, wobei Design und leise Komponenten ein wichtiges Auswahlkriterium sind. Hardware-Teile wie LC-Display und Infrarotempfänger werden in Abschnitt 4.3 beschrieben.

Kapitel 5 befasst sich mit den Software-Komponenten. In Abschnitt 5.1 wird die Entwicklung einer grafischen Benutzerschnittstelle beschrieben. Das dort entwickelte Toolkit wird vorgestellt, welches die Erstellung von Menüs, Knöpfen, Scrollbalken und Fortschrittanzeigen bietet. Der Aufbau der verwendeten NMM-Knoten, sowie die Entwicklung neuer Knoten wird in Abschnitt 5.2 beschrieben.

Aufbau, Funktionsweise und Konfiguration der Hauptanwendung (Multimedia-Box Anwendung) wird in Kapitel 6 beschrieben. Abschnitt 6.1 erläutert die Bedienung der Anwendung. In Abschnitt 6.2 wird die Architektur und Implementierung der Hauptanwendung beschrieben, die eine leichte Konfiguration bietet und so aufgebaut ist, dass sie leicht erweitert werden kann. Abschnitt 6.3 beschreibt die Implementierung aller integrierter Funktionen.

Abschließend befindet sich eine Zusammenfassung und mögliche Erweiterungen der Multimedia-Box in Kapitel 7.

1.3 Hinweis

Diese Arbeit führt die Entwicklung einer Multimedia-Box, die in einem Fortgeschrittenen-Praktikum [65] entstanden ist, fort. An diesem Praktikum habe ich aktiv mitgearbeitet, wobei schon Teile der vorliegenden Arbeit in die Praktikumsarbeit integriert wurden.

Diese Arbeit stützt sich auch auf Komponenten, die im Rahmen des Fortgeschrittenen-Praktikums entstanden sind. Auf Komponenten, die unverändert oder teilsweise daraus übernommen wurden, wird explizit hingewiesen.

Die verwendete Software, insbesondere die Netzwerk-Integrierte Multimedia Middleware, wird ständig weiterentwickelt. Die hier vorliegende Arbeit spiegelt den aktuellen Entwicklungsstand im Oktober 2003 wieder.

Kapitel 2

Verwandte Projekte und Systeme

Mittlerweile existieren viele verschiedene Programme, die alle Arten von Multimediadaten wiedergeben oder aufnehmen können, wenn die entsprechende Hardware, wie zum Beispiel eine digitale Satellitenkarte, vorhanden ist. Es gibt auch Bemühungen, den Funktionsumfang dieser Programme zu einem einzigen Programm zusammenzufassen.

In den folgenden Abschnitten werden vorhandene Projekte aufgeführt und mit der Multimedia-Box verglichen. Dabei werden Open Source-Projekte betrachtet, die sich auf die Entwicklung von Multimedia-Software-Komponenten beschränken. Aber auch Projekte, die neben der reinen Softwarelösung auf Hardware-Unterstützung zurückgreifen.

2.1 Open Source

Da kommerzielen Betriebssystem wie das Windows XP Mediacenter [59] von Microsoft oder Programme, wie z.B. der Windows Mediaplayer [60], der eine Vielzahl von Video- und Audiodateien mit den unterschiedlichsten Komprimierungscodecs abspielen kann, nicht im Quellcode vorliegen, können sie meist nur schwer erweitert werden. Im folgenden werden Programme vorgestellt, die im Quellcode vorliegen und sich somit schon leichter anpassen und erweitern lassen als Closed Source Programme.

2.1.1 MPlayer

Der MPlayer [72] ist ein Multimedia-Player für Linux Systeme, der verschiedene Audio- und Videodaten abspielen kann. Mit ihm lassen sich Audio-CDs abspielen, Video-DVDs betrachten und die unterschiedlichst komprimierten Audio- und Videodateien abspielen.

Der MPlayer lässt sich per Kommandozeile starten und lässt sich per Parameterübergabe anpassen und steuern. Optional kann er aber auch mit einem grafischen Benutzerinterface ausgestattet werden (vgl. Abbildung 2.1). Das Benutzerinterface

Abbildung 2.1: Das Benutzerinterface des Linux MPlayers (unten rechts) präsentiert sich auf dem Linux Desktop (Bildquelle aus [72]).

präsentiert sich auf dem Desktop, hier können einzelnen Funktionen wie File Öffnen, Abspielen und Spulen mit der Maus angeklickt und aktiviert werden.

Der Player liegt im Quellcode vor und lässt sich demnach beliebig anpassen und erweitern. Allerdings gestaltet sich die Erweiterung ziemlich schwierig, da der Player keine plugin-Struktur besitzt. Man braucht sehr gute Kenntnisse des Quellcodes, um den Player zu erweitern.

Ziel der Multimedia-Box ist u.a. die leichte Erweiterbarkeit. Sie besitzt eine plugin-artige Architektur, mit der leicht zusätzliche Funktionen integriert werden können (vgl. Abschnitt 6.2).

2.1.2 Alsaplayer

Der Alsaplayer 2.2 ist ein Opernsource-Player. Mit ihm können nur Audiodateien wiedergeben werden. Er kann Audio-CDs und komprimierte Audiodateien abspie-

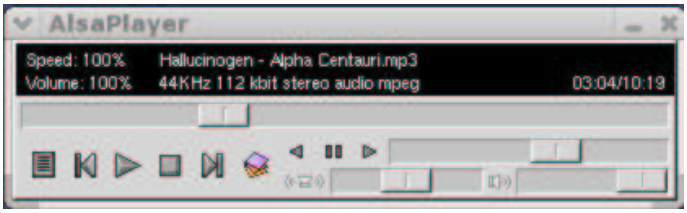

Abbildung 2.2: Benutzerinterface des Linux Alsaplayers

len. Abbildung 2.2 zeigt die Benutzerschnittstelle des Alsaplayers.

Eine besondere Funktion des Alsaplayers ist die Verwaltung von Playlisten. In eine Playlist können verschiedene Audiodateien aufgenommen werden und ihre Reihenfolge verändert werden. Wird eine Playlist zum Abspielen angewählt, so werden die enthaltenden Audiodateien in der gespeicherten Reihenfolge abgespielt.

Die Playlist wird im M3U-Format gespeichert. Eine M3U-Datei ist eine Textdatei, in der pro Zeile der zu spielende Titel steht. Das M3U-Format wird von vielen Player unterstützt, wie z.B. auch von dem Windows-Audioplayer Winamp [63]. Erweitert wird der Alsaplayer durch Plugins, die allerdings nur Audiodaten verarbeiten können.

Eine Anforderung an die Multimedia-Box ist, dass sie neben Audiodaten auch Videodaten bzw. Audio/Videodaten abspielen soll, um beispielsweise das Fernsehprogramm wiederzugeben. Die Multimedia-Box besitzt daher die Möglichkeit Videodaten, die in unterschiedlichen Formaten vorliegen können, wiederzugeben. Die Multimedia-Box unterstützt ebenfalls M3U-Dateien und Playlisten. Ihre Playliste wurde erweitert, so dass dort auch Videodateien oder Audio-CD Titel aufgenommen werden können.

2.1.3 GStreamer Player

Der GStreamer Player [29] basiert auf der GStreamer Middleware und ist modular aufgebaut. GStreamer erlaubt die Konstruktion von Multimedia-Graphen. Jeder Knoten dieses Graphs erfüllt eine bestimmte Funktion, wie z.B. das Lesen oder Dekodieren einer Audiodatei. Knoten können beliebig ausgetauscht oder erweitert werden. Somit kann man sehr leicht Multimedia-Player erstellen oder erweitern.

Die Multimedia-Box basiert auf der Netzwerk Integrierten Multimedia Middleware (NMM [62]), die einen ähnlichen Ansatz verfolgt. Kapitel 3 geht noch genauer auf den Aufbau von NMM und die Entwicklung von Multimedia-Applikationen ein. Grundlegende Erweiterung von NMM gegenüber GStreamer ist die Integration des Netzwerks in Multimedia-Anwendungen. Mit Hilfe von NMM ist es möglich, eine Multimedia-Anwendung auf mehrere Rechner im Netzwerk zu verteilen. Momentan noch nicht implementiert, aber denkbar ist die Vernetzung mehrerer Multimedia-Boxen, die sich rechenintensive Aufgaben teilen können.

Ziel der Multimedia-Box ist u.a. auch die leichte Integration neuer Funktionen oder Audio/Videoformate, die durch NMM vereinfacht wird, da sie modular aufgebaut ist und neue Module einfach in die Anwendung integriert werden können.

2.1.4 Video Disk Recorder

Der Video Disk Recorder (VDR) [46] ist ein Open Source Projekt, das sich mit dem Empfang und der Aufnahme von digitalem Fernsehen beschäftigt. Die digitalen Satellitendaten werden mit Hilfe einer oder mehrerer DVB-PCI-Karten empfangen.

What's on now? Sun 7.04 13:21
1 RTL 12:05 Griffelkin – Ein Teufel auf Abwegen
2 Sat 1 13:05 Die Drei
3 Pro-7 13:19 Wieder Ärger mit Bernie
4 RTL 2 12:30 Ferien im Action Camp
5 ARD 13:15 Weltreisen:Plaza Latina
6 BR3 13:15 Bayern erleben
7 Hessen 12:30 Die Kreuzzüge
8 N3 13:00 Der wahre Kir Royal
9 SR3 13:00 Tiergeschichten aus dem Meer
10 WDR 13:00 Prinz Bajaja
11 BR-alp 13:15 Erzähltes Land
12 SWR BW 13:00 Tiergeschichten aus dem Meer
13 Phoeni 13:00 Städtebilder Ost
14 ZDF 13:15 ZDF.umwelt
Record
Next
Schedule
Switch

Abbildung 2.3: Benutzerinterface des Video Disk Recorders

Dabei muss mindestens eine davon über einen Hardware-MPEG2-Decoder verfügen, da der VDR keinen Software-MPEG2-Decoder implementiert hat. Außerdem stützt sich das komplette Benutzer-Interface auf das von der DVB-Karte bereitgestellte On-Screen-Menü (OSD) (siehe Abbildung 2.3). Dieses OSD benutzt der VDR, um die Kanalliste, Uhrzeiten, Fortschrittsanzeigen etc. anzuzeigen. Bedient wird der VDR durch die Tastatur, kann aber auch per Infrarotfernbedienung gesteuert werden.

Für den VDR existieren verschiedene Patches, mit denen neben dem Fernsehprogramm auch z.B. Video-DVDs abgespielt werden können.

Eine Einschränkung entsteht durch die Verwendung des OSD der DVB-Karte. Es gibt DVB-Karten, die keinen TV-Ausgang besitzen und somit kein OSD unterstützen. Der VDR kann mit solchen DVB-Karten nicht betrieben werden.

Die Multimedia-Box verfolgt deshalb einen anderen Ansatz. Sie hat einen eigenen Mechanismus, um ihr Benutzer-Interface mit Hilfe einer Grafikkarte darzustellen und ist deshalb nicht auf das OSD der DVB-Karte angewiesen.

2.1.5 Vergleichbare Systeme

Es gibt Ansätze, um die Funktionalität der oben genannten Programme zu vereinen. ”The Linux Home-Entertainment-Server” [9] und ”Freevo” [25] sind zwei Systeme, die sich dieses Ziel gesetzt haben. Sie benutzen schon vorhandene Programme wie MPlayer oder VDR, um Mediadateien abzuspielen bzw. aufzuzeichnen. Sie implementieren eine grafische Benutzerschnittstelle und rufen je nach Benutzereingabe die verschiedenen Programme auf.

Nachteil dieses Konzeptes ist, dass das System auf die Funktionen der aufzu-

rufenden Programme beschränkt ist. Soll die Funktionalität erweitert werden, so muss u.a. das aufzurufenden Programm erweitert werden.

Einen flexibleren Ansatz, wie ihn die Multimedia-Box verfolgt, ist die Verwendung einer Multimedia-Middleware (vgl. Kapitel 3).

2.2 Ansätze aus der Forschung

2.2.1 SAMBITS

”System for Advanced Multimedia Broadcast and IT Services” (SAMBITS) [27] ist ein Projekt, das eine Architektur für die Übertragung und Empfang von Diensten definiert, die über Rundfunkt, Satellit und Internet bereitgestellt werden. Hauptziel dieses Projekts ist die Integration des MPEG4 und MPEG7 Standards in die Rundfunkt- und Satellitenübertragungstechnologie. Momentan werden über Satellit nur Videodaten im MPEG2 Format ausgestrahlt. Mit MPEG4 und MPEG7 will man den Zugriff auf Meta-Daten erreichen. Mit einer entsprechenden Settop-Box, auf der die SAMBITS Architektur implementiert ist, kann man dann auf webähnliche Inhalte zugreifen oder Datenbankabfragen starten. SAMBITS bietet keine Multimedia-Middleware, sondern verwendet die Middleware, die durch eine MHP [54] Implementierung (vgl. Abschnitt 2.3) bereitgestellt wird. Integriert wurden zusätzlich ein MPEG4 Player und eine MPEG7 Engine. Die Implementierung der Software wurde in Java realisiert, da die MHP Implementierung eine Java Schnittstelle besitzt. Als Hardwarebasis wird ein Settop-Box PC von Fujitsu-Siemens [26] benutzt.

Der Settop-Box PC von Fujitsu-Siemens könnte auch als Hardware-Grundlage für die Multimedia-Box dienen. Doch die Erweiterung eines solchen System ist schwierig. Auf Grund des kompakten Designs, lassen sich kaum Erweiterungskarten integrieren. Auch die Verfügbarkeit ist nicht gewährleistet. In zwei Jahren etwa wird der Settop-Box PC vielleicht nicht mehr hergestellt.

SAMBITS konzentriert sich auf Rundfunkt- und Satelliten-Dienste. Bestrebungen zur Integration von CD/DVD Spieler oder sonstigen Multimedia-Anwendungen wie es die Multimedia-Box anstrebt, findet man hier nicht.

2.2.2 Media/Entertainment Gateway

Einen Ansatz, den Intel [40] zusammen mit Sigma Designs [73], Focus Enhancement [23] und TUXIA [81] anstrebt, ist der Einsatz von speziell optimierter Hardware und Software für ein Home Entertainment System [39]. Dieses System soll alle im Wohnzimmer befindlichen Geräte wie Satellitenreceiver, DVD Player und Videorekorder ersetzen. Darüberhinaus soll das System auch als ”Communication Gateway” benutzt werden können. Es soll als Verbindungsstück zwischen Wohnung und der Außenwelt dienen. Angestrebt ist ein System, mit dem alle Geräte (bis hin zur Klimaanlage) verbunden sind und eine Verbindung zur Außenwelt hat.

Diese Verbindung kann eine normale Internetverbindung sein, aber auch durch eine Verbindung mit einer Satellitenanlage realisiert werden.

Bei der Zusammenstellung der Systemkomponenten will man einen alleinigen Kompromiss zwischen hochoptimierter Hardware und Software eingehen. Durch Einsatz von Spezialhardware wäre das System nicht flexibel genug, wohingegen die reine Softwarelösung einen schnellen Prozessor benötigen würde, der teuer ist und eine gute Kühlung braucht. Eingesetzt wird deshalb ein kostengünstiger Intel Celeron Prozessor (700 Mhz). Da dieser Prozessor nicht genug Performance bietet, um die oben genannten Anforderungen alle zu erfüllen, bzw. nicht gleichzeitig zu erfüllen, muss das System mit Hardware ausgestattet werden, die die CPU entlastet.

Softwareseitig basiert das System auf der Linuxdistribution TASTE, die von TUXIA entwickelt wird. TASTE ist eine embedded Linuxdistribution mit einem embedded Internetbrowser. Genauere Informationen über die Software stehen nicht zur Verfügung.

Ziel der Multimedia-Box ist es sowohl Hardware als auch Softwarelösungen zu unterstützen. Das bedeutet beispielsweise, dass ein MPEG komprimiertes Video abgespielt werden kann, indem es mit einem Software-MPEG-Dekoder dekodiert wird. Steht entsprechende Hardware zur Verfügung, um diese Aufgabe zu erledigen, so soll auch diese optional genutzt werden können.

2.2.3 KOM Player

Im Projekt “KOM-Player” [58] wird eine Plattform entwickelt, mit der die Entwicklung von Video-on-demand Anwendungen unterstützt werden soll. Die Übertragung von Audio/Video Daten wird über das Internet realisiert. Motivation für die Entwicklung einer solchen Plattform ist, dass bestehende kommerzielle Systeme, die Audio/Video Übertragungen (Streaming) über das Internet realisieren, nicht die Übertragungsqualität besitzen, wie die einer TV Übertragung.

Das System besteht aus drei Teilen: Client, Server und Proxy-Cache. Momentan werden die Übertragungsprotokolle RTP/RTCP [32], RTSP [33] und SDP [45] unterstützt. Client und Server sind aus mehreren Modulen aufgebaut. Der Client besitzt ein Empfangsmodul, das Daten vom Server empfangen kann, ein Dekodermodul, das die komprimierten Daten dekodiert und ein Anzeigemodul, das die dekodierten Daten in einem Fenster ausgibt. Die Audio/Video Daten sind im MPEG1 Format kodiert.

Der Server verschickt die MPEG1 kodierten Daten an die Clients. Der Proxy-Cache speichert zusätzliche Daten, damit RTSP Anfragen verarbeitet werden können.

Die KOM-Player Plattform ist als Grundlage für die Entwicklung von Streaming-Anwendungen übers Internet gedacht. Ein Ziel der Multimedia-Box ist jedoch den Zugang zu vielen unterschiedlichen Datenquellen bereitzustellen, beispielsweise der Empfang von (digitalen) TV Programmen über Kabel/Satellit oder Wiedergabe einer CD oder DVD.

2.2.4 Interaktives TV System

Das interaktive TV System (ITV) ist ein verteiltes System, dessen Ziel es ist, einem Endbenutzer, der wenig Erfahrung mit Computern hat, verschiedene Dienste anzubieten [56]. Neben dem Empfang von TV Programmen werden auch Anwendungen wie Spiele oder Home-Shopping angeboten.

Das System besteht aus mehreren Servern, die den Clients (Settop-Boxen) ihre Dienste über ATM anbieten. Das Betriebssystem der Server ist IRIX, eine UNIX Version von Silicon Graphics. Die Settop-Boxen laufen mit einem speziell angepassten Echtzeit-Kernel. Genaue Informationen über die Hardware-Komponenten insbesondere der Settop-Box stehen nicht zur Verfügung. Die Settop-Box besitzt keine Festplatte oder CD/DVD Laufwerk. Gesteuert wird sie über eine Fernbedienung, mit der einzelne Kanäle angewählt werden. Diese Kanäle repräsentieren die einzelnen Anwendungen, wie z.B. Home-Shopping oder Spiele.

Grundlage der Systems bildet das Objekt Communication System (OCS). Neben der Implementierung von verteilten Objekten bietet es u.a. eine Authentifikation, so dass bestimmte Dienste für einen Client freigegeben, bzw. gesperrt werden können. Implementiert ist auch ein “Resource Audit Service”, der Nachrichten über den Zustand des Systems verschickt, etwa wenn Systemkomponenten nicht richtig funktionieren oder nicht vorhanden sind.

Die ausgewählt Anwendung wird vom Application Manager (AM) in den Speicher der Settop-Box kopiert und gestartet. Die Anwendung liegt als ausführbarer Code vor und wird direkt ausgeführt.

Da die Settop-Box keine Festplatte besitzt, können empfangene TV-Sendungen nicht aufgenommen werden, wie bei der Multimedia-Box. Das System ist auf Dienste angewiesen, die von den Server bereitgestellt werden. Funktionen wie DVD oder CD-Spielen, die auch ohne Server durchgeführt werden können, werden nicht unterstützt. Ein ähnliches System wurde auch von Microsoft entwickelt [55]. Dieses System benutzt einerseits andere Hardwarekomponenten, andererseits enthält das verwendete Betriebssystem Teile des Windows Systems.

2.2.5 CustomTV

Die ständig wachsende Anzahl von Diensten, die über Kabel oder Satellit angeboten werden, erfordern ein neues Design der Benutzerschnittstelle. Bisher beschränkte sich das Benutzerinterface auf die Auswahl von Fernsehkanälen. In [67] wird ein neues Konzept der Benutzerschnittstelle speziell für den TV-Bereich vorgestellt. Dieses Konzept wird allerdings auch bei der Gestaltung des Benutzerinterface der Multimedia-Box verwendet.

Alle Interaktionen werden mit Hilfe einer Fernbedienung durchgeführt. Das Benutzerinterface ist so gestaltet, dass die Technologie oder die Implementierung dem Benutzer verborgen bleibt. In [67] werden z.B. keine Unterschiede gemacht, ob der Benutzer einen MPEG-2 oder MPEG-4 Videostrom empfängt. Auch die Bedienung der Multimedia-Box erfordert kein Wissen über die abzuspielenden For-

mate. Es können so z.B. auf Knopfdruck verschiedene Dateien abgespielt werden, die unterschiedliche Formate besitzen (vgl. Abschnitt 6.1.7).

Das in [67] beschriebene System verwaltet auch benutzerspezifische Vorlieben, wie z.B. eine nach Interesse sortierte Senderliste. Die Multimedia-Box bietet die Möglichkeit, das Aussehen der Anwendung ganz individuell zu gestalten. Menügrafiken können durch andere ersetzt werden und neu angeordnet werden.

2.2.6 TV Anytime Forum

Das TV Anytime Forum [3] ist ein Zusammenschluss mehrerer Organisationen, die verschiedene Spezifikationen erstellen, um Audio- und Videoanwendungen und andere Dienste auf Konsumergeräten zu ermöglichen. In [71] wird die Verwendung von lokalen Speichergeräten untersucht, um TV Anytime Dienste zu implementieren.

Da es immer mehr Programmanbieter gibt, steigt auch die Programmvielfalt, die einem Endbenutzer zur Verfügung steht. Lokale Speichergeräte, wie z.B. eine Festplatte, ermöglichen es, dass der Verbraucher die Programme aufzeichnen kann und besitzt dadurch die Möglichkeit, sie genau dann anzuschauen, wenn er Zeit dafür hat [21]. Auch die Auswahl der aufzuzeichnenden Programme wird auf Grund der Vielfalt immer schwieriger. Das STORit-Projekt [2] verfolgt das TV Anytime Konzept. Die STORit Box bezieht Programme über Satellit, Kabel oder Internet. Ein integrierter digitaler Videorekorder speichert die Programme auf die lokale Festplatte. Bedient wird die Box über eine Fernbedienung und ein grafisches Benutzerinterface. Der Kern des Systems ist der “Content Manager”. Er verwaltet alle Metadaten, wie Aufnahmeliste, elektronischer Programmführer (EPG) und steuert die Wiedergabe und Aufnahme der Programme. Durch Selektion der EPG Daten können Aufnahmen direkt aufgenommen oder programmiert werden.

Die Multimedia-Box bietet ebenfalls die Möglichkeit Fernsehprogramme zeitgesteuert aufzunehmen. Momentan werden Aufnahmen durch manuelle Eingaben programmiert, geplant ist aber auch eine EPG basierte Programmierung.

2.3 Multimedia Home Plattform

Die Multimedia Home Plattform (MHP) [54] bietet eine einheitliche Schnittstelle für die Entwicklung von interaktiven, multimedialen Diensten für digitale TV-Programme. MHP-Anwendungen sind zum Beispiel News Ticker, Game-Shows oder ein elektronischer Programmführer. Es gibt auch MHP-Anwendungen wie zum Beispiel Home-Shopping, die einen Rückkanal verwenden (über Modem oder Netzwerk).

MHP bietet eine Java Programmierschnittstelle an, mit der Anwendungen für eine Settop-Box geschrieben werden können. Die MHP Implementierung muss an gegebene Hardwareverhältnisse angepasst werden. Das Institut für Rundfunktechnik [36] hat eine Referenzimplementierung für einen Standard-PC veröffentlicht,

jedoch ist zum jetzigen Zeitpunkt noch keine Linux-Version verfügbar. Java ist zwar Plattform-unabhängig, jedoch greifen die Java-Module auf die Hardware, wie z.B. eine DVB-Karte zu. Diese Module müssen an Betriebssystem und Hardware angepasst werden.

Im Vordergrund stehen bei MHP digitale TV Dienste. In wie weit die Entwicklung von Funktionen wie man sie von den Multimedia-Playern her kennt in späteren Versionen voranschreitet, steht noch nicht fest. Bis dahin ist MHP für reine Settop-Boxen sicher eine gute Lösung, doch für eine Multimedia-Box noch zu begrenzt, da es z.B. noch keine Möglichkeit gibt einen DVD-Spieler zu integrieren oder Videodaten in andere Formate zu konvertieren.

Kapitel 3

Die Netzwerk-Integrierte Multimedia Middleware (NMM)

Die “**N**etzwerk-Integrierte **M**ultimedia **M**iddleware” (NMM) wird am Lehrstuhl für Computergrafik der Universität des Saarlandes entwickelt. Weiterführende Informationen über das Projekt und seinen aktuellen Entwicklungsstand findet man im Internet [62]. NMM ist ein Open Source C++ Framework, das die Entwicklung von Multimedia-Software unter Linux ermöglicht [11]. Ein Ziel von NMM ist es, den Zugang zu den verschiedensten Hardwarekomponenten und einer vielzahl von Multimediaformaten zu ermöglichen.

Das Schlagwort Multimedia-Applikation umfasst eine weitreichende Palette von Anwendungen. Dabei spielt vorallem die Integration von Audio und Video eine wichtige Rolle. Beispiele für Multimedia-Anwendungen sind CD-Spieler, DVD-Spieler, rechnerübergreifende Videokonferenzsysteme oder digitales Fernsehen, mit all seinen Möglichkeiten wie Video-On-Demand oder Auswahl verschiedener Perspektiven, d.h. auch die Interaktion mit dem Benutzer spielt eine wichtige Rolle.

Die Entwicklung von Multimedia-Anwendungen benötigt Kenntnisse über Geräteeigenschaften wie etwa einer Kamera, DVD-Laufwerk oder Grafikkarte. Diese Geräte müssen gesteuert werden und auch miteinander verbunden werden. Zum Abspielen einer Video-DVD reicht es allerdings nicht aus, wenn man das DVD-Laufwerk mit der Grafikkarte und Soundkarte verknüpft. Die Daten, die von der DVD gelesen werden, müssen erst in die für Sound- und Grafikkarte “verständliche” Daten umgewandelt werden.

Die NMM-Middleware hilft dem Multimedia-Anwendungsprogrammierer dabei, die Anwendung modular zu gestalten (Plugin-Architektur) und Geräte und Datenumwandlungen zu abstrahieren, wie z.B. das Auslesen einer DVD oder die Dekodierung von komprimierten Videodaten. Geräte und Softwarekomponenten werden in einen *Multimedia-Datenflussgraphen* aufgenommen, der den Fluss des Datenstroms definiert.

Die Middleware ist auch für die *Synchronisation* von Audio- und Videodaten

verantwortlich, d.h. sie kontrolliert den Datenfluss so, dass Bild und Ton synchron wiedergegeben werden [74].

Im NMM-Projekt spielt das Netzwerk eine zentrale Rolle [51]. Dadurch ist es möglich, Geräte, die nicht am eigenen Computer angeschlossen sind zu kontrollieren. Das Netzwerk ist dabei transparent für die Anwendung, d.h. entfernte Geräte, die über das Netzwerk kontrolliert werden, werden genau so angesteuert wie lokale Geräte. Eine rechenintensive Anwendung kann auch auf mehrere Rechner verteilt werden.

3.1 Übersicht

Die Hauptelemente von NMM sind *Knoten* (engl. Nodes), *Ein- und Ausgangs-Jacks* und *Nachrichten* (engl. Messages).

- Die Knoten sind die funktionalen Elemente von NMM. Im Allgemeinen verkörpert ein Knoten ein physikalisches Gerät z.B. CD-Laufwerk oder Grafikkarte, oder er repräsentiert eine bestimmte Funktion, z.B. kann er Bilddaten manipulieren oder Dateioperationen ausführen. Mehrere Knoten werden zu einem *Flussgraphen* zusammengefasst. Der Graph kann sich komplett auf einem Rechner befinden oder Teile des Graphen können sich jeweils auf anderen Rechnern befinden, wobei die Verteilung des Graphen für die Anwendung transparent ist.

- Jacks sind die Ein- und Ausgänge der Knoten. Sie bauen die Verbindung zwischen anderen Knoten auf. Ihre Aufgabe besteht in der Annahme (Input-Jack) und dem Weiterreichen (Output-Jack) von Nachrichten. Mit jedem Jack sind automatisch ein oder mehrere *Formate* verbunden. Formate beschreiben, welche Art von Daten ein Knoten letztendlich verarbeiten kann.

- Die Nachrichten repräsentieren verschiedene Informationsarten, die durch den Graph fließen können. Das können einerseits reine Multimediadaten sein (Buffer), andererseits können es Informationen oder Steuerbefehle, sogenannte Events, also Ereignisse sein, die der Kommunikation zwischen Knoten dienen.

3.2 Flussgraph

Abbildung 3.1 zeigt eine detaillierte grafische Darstellung eines Knoten. Jeder Knoten kann keinen, einen oder mehrere Ein- oder Ausgänge haben, dabei wird jeder Ein- oder Ausgang durch einen Jack repräsentiert. Damit mehrere Knoten zu einem Graphen verbunden werden können, müssen ihre Ausgangs-Jacks, die ein bestimmtes Format besitzen mit den Eingangs-Jacks verbunden werden, die das selbe Format besitzen.

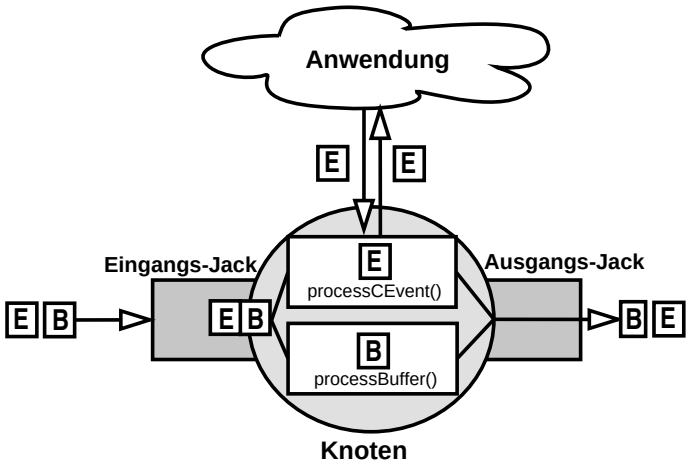

Abbildung 3.1: Ein NMM Knoten mit in-stream Events, Buffern und out-of-band Events, die von oder zu der Anwendung geschickt werden. (”E” Event, ”B” Buffer). Buffer werden innerhalb der Knoten verarbeitet und Events werden mit registrierten Methoden verarbeitet, die Teil des Knoten oder der Anwendung sein können.

Knoten werden zu einem Flussgraphen verbunden. Ein Knoten empfängt Daten von seinem Vorgängernode, verarbeitet sie und sendet die verarbeiteten Daten zu seinem Nachfolgernode, d.h. die Daten im Graphen fließen flussabwärts (engl. downstream). Es gibt aber auch die Möglichkeit, bestimmte Nachrichten flussaufwärts zu verschicken (engl. upstream).

Die `processBuffer( Buffer* )` Methode wird immer dann aufgerufen, wenn neue Daten am Knoteneingang vorliegen. Der mitgelieferte Buffer enthält die empfangenen Daten und deren Größe. Diese Methode ist für eigentliche Verarbeitung der Daten verantwortlich.

Handelt es sich bei den empfangenen Daten um ein Event, dann wird die `processCEvent` Methode aufgerufen, die eine für das Event registrierte Handler-Methode aufruft. Events, die durch den Flussgraphen fließen werden *in-stream Events* genannt.

Auch die Anwendung selbst kann sich für Events registrieren oder auch Events an einen Knoten schicken. Solche Events werden *out-of-band Events* genannt. Ein Objekt, das Events empfangen kann, wird als *Listener* bezeichnet.

Abbildung 3.2 zeigt einen einfachen NMM Graph mit drei Knoten für einen MP3 Player. Der `MP3ReadNode` liest eine MP3 Datei von der Festplatte ein und schickt die eingelesenen Daten über seinen Ausgangs-Jack an seinen Nachfolgerknoten, den `MPEGAudioDecodeNode`. Dieser Knoten empfängt die Daten und dekodiert sie. Die dekodierten Audiodaten werden dann zum `PlaybackNode` geschickt. Wenn die Audiosamples am `PlaybackNode` ankommen, werden sie dort

an die Soundkarte geleitet und wiedergegeben.

3.2.1 Formate

Unter einem Formate versteht man im allgemeinen die beschreibenden Metadaten eines Multimedia-Datenstroms zwischen den NMM-Knoten [68, 53]. Diese Metadaten beinhalten Eigenschaften der Datenströme wie zum Beispiel die *Auflösung* oder *Bildwiederholrate* eines Videostroms. Ein Beispiel für verschiedene Auflösungen und Bildwiederholraten ist ein Video-DVD. Bei Videotiteln im *PAL* Format ist die Auflösung höher als bei Titeln im *NTSC* Format. Jedoch ist die Bildwiederholrate bei *NTSC* etwas höher.

Im NMM-Projekt sind die Formate hierarchisch eingeteilt. Es gibt verschiedene Typen, wie etwa Audio, Video oder Text, die wiederum mehrerer Untertypen haben können (vgl. Formatklassifikation [68]). Die Untertypen sind wichtig, da durch die grobe Einteilung wie etwa Audio das Format noch nicht genau bestimmt ist. Untertypen könnten hier verschiedene Audiokompressionstypen sein.

Die genauen Eigenschaften eines Formates werden als *Parameter* bezeichnet. Ein Parameter besteht aus einem eindeutigen Namen, dem Parameternamen und einem Wert, dem Parameterwert [68]. Knoten können mehrere Ein- oder Ausgangsformate besitzen. Alle unterstützen Formate eines Knotens werden in den *Properties* zusammengefasst.

Ein Knoten kann mit der Methode `getOwnInputProperty()` ein Property-Objekt für seinen Eingang anfordern oder mit `getOwnOutputProperty()` ein Objekt für seinen Ausgang. Das Objekt besitzt u.a. die Methode `addNewFormat`, um ein neues Format hinzuzufügen. Der Rückgabewert dieser Methode ist ein Format-Objekt, dem Parameter hinzugefügt werden können.

Als Beispiel wird hier das Ein- und Ausgangsformat eines Videodekoders spezifiziert:

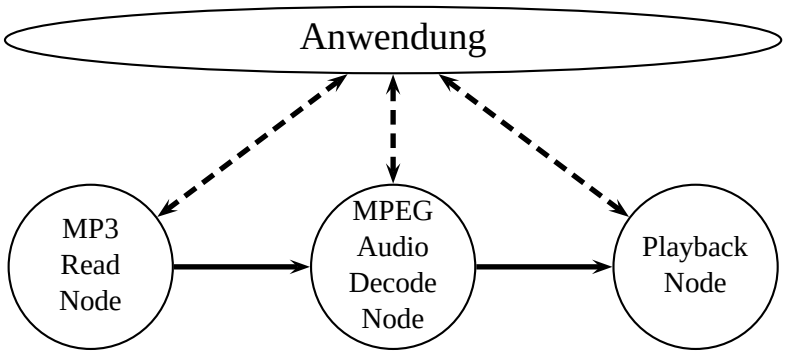

Abbildung 3.2: Ein einfacher MP3 Player, realisiert mit Hilfe des NMM Frameworks.

```
/* Eingangsformat hinzufügen */
getOwnInputProperty()->addNewFormat("video/mpeg2");

/* Ausgangsformat hinzufügen */
Format* i_format = getOwnOutputProperty()
              ->addNewFormat("video/raw");

/* Bildauflösung */
i_format->addWildcard("x_resolution");
i_format->addWildcard("y_resolution");

/* Reihenfolge der Planes */
i_format->addStringParamValue("channelorder","yvu");

/* Farbformat */
i_format->addStringParamValue("colorspace","yv12");

/* Bits pro Pixel */
i_format->addIntParamValue("bitperpixel",12);

/* Anordnung der Pixel */
i_format->addStringParamValue("format", "planar");
```

Das Eingangsformat des Videodekoders ist “video/mpeg2”, d.h. der Typ ist “video” und der Untertyp “mpeg2”. Der Knoten akzeptiert also alle im mpeg2 Format komprimierten Videodaten.

Das Ausgangsformat ist “video/raw”, d.h. der Knoten liefert unkomprimierte Einzelbilder. Das Format der Einzelbilder wird durch die Parameter genauer definiert. Parameterwerte können Texte, Integer- oder Fließkommazahlen sein und werden je nachdem mit `addStringParamValue`, `addIntParamValue` oder `addFloatParamValue` hinzugefügt. Parameter, die zu Beginn noch nicht festgesetzt werden können, weil sie erst mit Hilfe der eingehenden Daten gesetzt werden können, werden mit der Methode `addWildcard` hinzugefügt. In obigem Beispiel ist das die Bildauflösung, da sie für einen bestimmten Datenstrom erst durch den Videodekoder bestimmt werden kann.

3.2.2 Knoteneinteilung

Nachfolgende Aufzählung listet alle Arten von Knoten mit ihren unterschiedlich vielen Ein- und Ausgängen auf.

Quellknoten Ein Quellknoten (engl. source node) hat keine Eingangs-Jacks und einen Ausgangs-Jack. Er ist ein Knoten, der nur Daten erzeugen und ausgeben kann. Im Beispiel des MP3 Players (Abbildung 3.2) ist der MP3Read-Node ein solcher Quellknoten.

Senkeknoten Ein Senkeknoten (engl. sink node) hat einen Eingangs-Jack und keine Ausgangs-Jacks. Ein Senkeknoten kann nur Daten annehmen. Der Play-

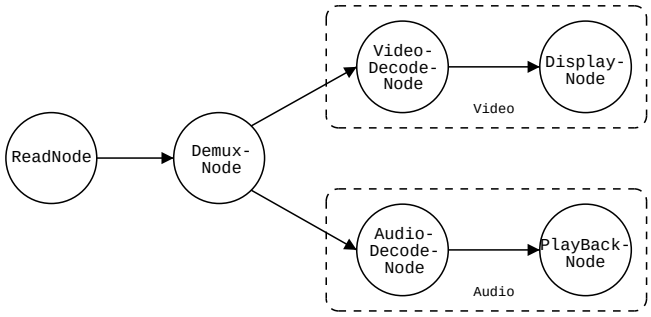

Abbildung 3.3: Ein Flussgraph, der ein Audio/Video-Datei wiedergibt, realisiert mit Hilfe des NMM Frameworks.

backNode aus dem Beispiel in Abbildung 3.2 ist ein Senkeknoten.

Verarbeitungsknoten Ein Verarbeitungsknoten (engl. processor node) hat einen Eingangs-Jack und einen Ausgangs-Jack. Der Knoten liest Daten von seinem Eingang, verarbeitet sie und schickt die verarbeiteten Daten an seinen Ausgang. Abhängig von dem Ein- und Ausgangsformt können zwei Untertypen unterschieden werden: Konverter und Filter.

Ein Konverterknoten liest Daten, die in einem bestimmten Format vorliegen, von seinem Eingang und wandelt sie in ein anderes Ausgangsformat um. Der MPEGAudioDecodeNode im MP3 Player Beispiel (Abbildung 3.2) ist ein Konverterknoten. Er wandelt die einkommenden komprimierten Audiodaten in unkomprimierte Audiodaten um, die mit dem PlaybackNode abgespielt werden können. Ein Konverterknoten kann auch zur Umwandlung von Videodaten benutzt werden, etwa die Konvertierung eines RGB kodierten Bildes in das YUV Farbformat.

Die Jacks eines Filterknoten haben das gleiche Eingangsformat wie Ausgangsformat. Filterknoten können zum Beispiel in der Bildverarbeitung eingesetzt werden, um die Helligkeit eines Videobildes zu reduzieren oder erhöhen, dabei wird das eigentliche Videoformat nicht verändert.

Multiplexer-Knoten Wenn ein Knoten mehrere Eingänge und nur einen Ausgang besitzt, bezeichnet man ihn als Multiplexer-Knoten. Ein Multiplexer-Knoten kann Daten von all seinen Eingängen lesen, um daraus einen einzigen Ausgangsstrom zu generieren. Ein Beispiel für diese Art von Knoten ist ein OverlayNode, der zwei Videoströme einliest und daraus einen einzigen Videostrom generiert, indem er beide Eingangsvideobilder übereinander legt.

Demultiplexer-Knoten Das Gegenteil eines Multiplexer-Knoten ist der Demultiplexer-Knoten. Dieser Knoten teilt einen Eingangsstrom in mehrere Einzelströme auf. Ein Beispiel für einen solchen Knoten sieht man in Abbil-

dung 3.3. Diese Darstellung zeigt einen einfachen Flussgraphen zur Wiedergabe von Audio/Video-Daten, die in einer Datei gespeichert sind.

Dateien, die sowohl Audio- und Videodaten enthalten, enthalten diese in gemultiplexter Form, d.h. der Datenstrom enthält abwechselnd Audio- und Videodaten. Aufgabe eines Demultiplexers ist, diese gemultiplexten Daten wieder zu trennen.

Der Flussgraph besteht aus mehrerer Knoten. Der ReadNode liest eine Audio/Video-Datei von Festplatte und schickt die gelesenen Daten an den DemuxNode. Der DemuxNode teilt den einkommenden Datenstrom in seinen Videoteil und seinen Audioteil auf. Der Videoteil wird an den VideoDecodeNode geschickt, der die komprimierten Videodaten dekodiert. Der Audioteil wird an den AudioDecodeNode geschickt, wo sie dekodiert werden. Die dekodierten Daten werden dann an ihre entsprechenden Senkeknoten geschickt. Der DisplayNode präsentiert die dekodierten Einzelbilder und der PlaybackNode spielt die Audiodaten ab.

Zusammengesetzte Knoten Ein zusammengesetzter Knoten wird in NMM *CompositeNode* genannt [66]. Er ist ein Container für mehrere verschiedene Knoten. Er verhält sich für die Anwendung wie ein normaler NMM-Knoten. Die Anzahl der Ein- und Ausgänge ist anfangs noch nicht festgesetzt, erst wenn Knoten zum Composite Node hinzugefügt werden, stehen Anzahl und mögliche Format der Ein- und Ausgänge fest. Ein Beispiel für einen Composite Node zeigt Abbildung 3.4. Hier wird ein MP3ReadNode und MPEGAudioDecodeNode zu einem einzigen Composite Node zusammengefasst. Der Composite Node hat als Ausgangsformat das Format des MPEGAudioDecodeNode und kann direkt an den PlaybackNode angeschlossen werden.

3.3 Nachrichten

Innerhalb der NMM Architektur wird die Kommunikation durch ein einheitliches Nachrichtensystem realisiert. Es gibt zwei Arten von Nachrichten-Typen (engl. messages).

Alle Daten, die von Knoten generiert werden, werden in Nachrichten vom Typ `Buffer` gespeichert. Buffer werden entlang der Jacks von Knoten zu Knoten weitergereicht. Immer dann, wenn ein Buffer an einem Jack ankommt, wird er in eine Eingangsqueue eingereiht bevor er letztendlich zu der `processBuffer`-Methode des Knoten gelangt (vgl. Abschnitt 3.2).

Der zweite Nachrichten-Typ heißt `CEvent`. Events steuern das Verhalten eines Knotens oder liefern Informationen zur Anwendung. Das `CEvent` selbst ist eine Abkürzung für *Compound Event*, d.h. es ist aus mehreren einzelnen Events aufgebaut. Im folgenden wird der Begriff Event auch für das `CEvent` verwendet.

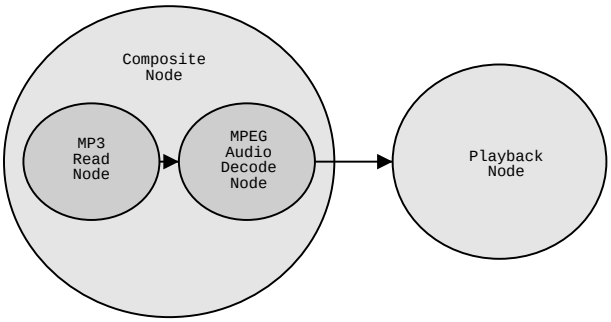

Abbildung 3.4: Ein zusammengesetzter Knoten, der einen MP3ReadNode und einen MPEGAudioDecodeNode enthält. Für die Anwendung verhält sich der Knoten wie ein normaler NMM-Knoten, der das Ausgangsformat des MPEG-AudioDecodeNode hat. Er lässt sich somit direkt mit einem PlaybackNode verbinden.

Es gibt zwei verschiedene Event-Typen. Der erste Typ wird *in-stream* Event genannt. Er wird wie ein Buffer behandelt, also an Nachfolgerknoten weitergereicht und in eine Queue eingereiht. Ein Beispiel dafür ist das *start_track* Event, das vom MP3ReadNode geschickt wird, um seinen Nachfolgern mitzuteilen, dass nun Daten einer neuen Datei folgen.

Der zweite Event-Typ heißt *out-of-band* Event. Diese Events werden von der Anwendung direkt an einen Knoten geschickt, um spezielle Parameter zu setzen oder auch zu empfangen. Ein Beispiel ist das Setzen des Dateinamens in einem Quellknoten. Das Setzen und Empfangen von Parametern wird durch Interfaces [51] erleichtert, die jedoch zum Zeitpunkt, als diese Arbeit erstellt wurde, noch nicht zur Verfügung standen.

Jedes Event besteht aus einem Schlüsselwert und zusätzlichen Werten. Der Schlüsselwert ist ein Name, der das Event eindeutig identifiziert. Damit ein Knoten überhaupt bestimmte Events empfangen kann, muss er sich vorher für diese registrieren und eine Methode bereitstellen, die das Event verarbeitet (sogenannte Handler-Methode). Ein Event-Dispatcher ruft diese Methode automatisch auf, wenn ein Event vom Knoten empfangen wird (Abbildung 3.1). Weitere Informationen über den Event-Mechanismus kann man in [57] finden.

Buffer und (in-stream) Events werden bei Knoten mit mehreren Ausgängen unterschiedlich behandelt. Besitzt ein Knoten mehrere Ausgänge, so wird ein ausgehender Buffer nur an einen vorher festgelegten Ausgang geschickt. Ein Beispiel dafür ist ein Demultiplexer-Knoten (siehe Abschnitt 3.2.2).

Events hingegen werden immer an alle Ausgänge geschickt, da nicht festgestellt werden kann, welche nachfolgenden Knoten sich für ein Event registriert haben. Ein Knotenprogrammierer hat aber die Möglichkeit, Events zu löschen und kann somit Events auf bestimmte Ausgänge eingrenzen.

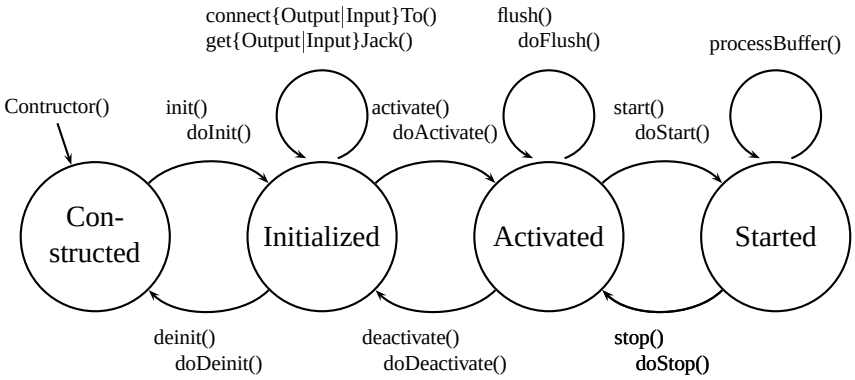

Abbildung 3.5: Zustände und Zustandsübergänge von NMM Nodes.

3.4 Speicherverwaltung

Eine mit dem NMM Framework entwickelte Multimedia-Anwendung besteht aus mehreren Knoten. Diese Knoten produzieren und löschen ständig Buffer. Der in Abbildung 3.3 gezeigte DemuxNode bekommt von seinem Vorgänger einen Buffer zugeschickt. Dieser Buffer wird auf Audio- und Videobestandteile überprüft. Die Audio- und Videoteile werden in jeweils getrennten Buffern abgelegt. Wurden alle Audio- und Videoteile des empfangenen Buffers in neue Buffer kopiert, so wird er gelöscht. Buffer werden also sehr schnell gelöscht und wieder neu angelegt. Das Löschen und Anlegen von Buffern braucht relativ viel Zeit. Eine zeitsparendere Strategie ist es, die Buffer nicht zu löschen, sondern einfach als “nicht benutzt” zu markieren. Wird ein neuer Buffer angefordert, wird ein nicht benutzter Buffer einfach wiederverwendet.

Aufgabe der Buffer-Manager ist die Verwaltung von Speicher, der effizient wiederverwendet werden kann. Jeder Knoten bekommt seinen Buffer von einem Buffer-Manager, der von NMM zur Verfügung gestellt wird. Dieser Buffer-Manager verwaltet Buffer und fordert sich vom Betriebssystem Speicherbereiche an. Ein Knoten kann sich mit

```
Buffer* buffer = getNewBuffer(Groesse in Bytes)
```

einen Buffer vom Buffer-Manager anfordern. Mit `buffer->release()` kann man einen nicht mehr benutzten Buffer wieder freigeben.

3.5 Die Entwicklung von NMM Knoten

Die verschiedenen Knotentypen (vgl. Abschnitt 3.2.2) sind von der Basisklasse `Node` abgeleitet und haben ein internes Zustandsmodell. Die Zustände und ihre zugehörigen Abhängigkeiten werden in Abbildung 3.5 gezeigt.

Die abgeleiteten Klassen repräsentieren die verschiedenen Knotentypen wie `GenericSourceNode` für den Quellknoten oder `GenericProcessorNode` für

den Verarbeitungsknoten. Ein Knotenprogrammierer leitet seinen Knoten von einer dieser Klassen ab und implementiert die Funktionalität dieses neuen Knotens, d.h. er implementiert die `processBuffer()` Methode, in der die eigentliche Verarbeitung der ankommenden Daten durchgeführt wird und bestimmt die Handler-Methoden, die aufgerufen werden sollen, wenn ein bestimmtes Event empfangen wird. Desweiteren kann er die unten aufgeführten Methoden implementieren, um bei Zustandübergängen bestimmte Aktionen auszuführen.

Folgende Methoden eines Knotens bewirken einen Zustandsübergang (Ausnahme: `flush`):

1. `init()`
2. `activate()`
3. `start()`
4. `stop()`
5. `deactivate()`
6. `deinit()`
7. `flush()`

Intern rufen diese Methoden folgende Methoden auf:

1. `doInit()`
2. `doActivate()`
3. `doStart()`
4. `doStop()`
5. `doDeactivate()`
6. `doDeinit()`
7. `doFlush()`

Diese Methoden werden Template-Methoden genannt (vgl. Template-Design-Pattern [22]). Sie ermöglichen die Vorgabe eines Gerüstes für einen bestimmten Algorithmus, wobei einige Schritte des Algorithmus' durch die Template-Methoden verändert werden können. Die oben aufgeführten Methoden sichern z.B. den neuen Zustand, nachdem die entsprechende Template-Methode aufgerufen wurde.

Alle obigen Template-Methoden müssen ein `Result` zurückliefern. Wenn eine Template-Methode ein `Result` zurückliefert, das nicht `SUCCESS` lautet, dann wirft die zugehörige Zustandsübergangsmethode eine Exception.

Folgende Methoden können von einem Knotenprogrammierer implementiert werden:

`Konstruktor` Im `Konstruktor` sollte der Knoten all seine Variablen auf definierte Werte setzten. Resourcenreservierungen, sollten hier nicht untergebracht werden, da sie vom Format abhängig sind. Die Registrierung von Events sollte hier durchgeführt werden (siehe Abschnitt 3.3).

`Result doInit()` Wenn die `doInit()` Methode aufgerufen wird, sollte der Knoten alle seine Ein- und Ausgangsformate festlegen und seine vom Format abhängigen Ressourcen reservieren.

`Result doActivate()` Innerhalb dieser Methode sollten alle Ressourcen reserviert werden, die für seine Ein- und Ausgangsformate benötigt werden.

	MP3ReadNode	MPEGAudio-DecodeNode	PlaybackNode
`Konstructor`	Alle Variablen werden auf definierte Werte gesetzt.		
`doInit()`	Öffne Datei, die im `CONSTRUCTED` State gesetzt wurde, prüfe Dateiformat, setze Ausgangsformat	Öffne MPEG Decoder-Library, setze Formate	Öffne Sounddevice, das im `CONSTRUCTED` State gesetzt wurde, prüfe Fähigkeiten der Hardware
`doActivate()`	—	Setze Parameter der Decoder-Library, abhängig von gewählten Formaten	Setze Parameter des Sound-Devices
`processBuffer()`	Sende start Event, lies Buffer aus der Datei	Dekodiere eingehende Buffer, verschicke dekodierte Daten	Schreibe eingehende Daten aufs Sound-Device
`doDeactivate()`	—		
`doDeinit()`	Schließe Datei	Schließe Library	Schließe Device
`doFlush()`	—	Setze Library zurück	Leere Audiobuffer

Tabelle 3.1: Beispiele für einige Methoden des MP3 Player Beispiels, das in Abbildung 3.2 gezeigt wurde.

`Result doStart()` Nach dem Aufruf der `start()` Methode, beginnt der Knoten, Nachrichten zu verarbeiten und weiterzuleiten (siehe Abschnitt 3.3). Ein Knotenprogrammierer muss normalerweise die `doStart()` Methode nicht implementieren.

`Message* proccessBuffer(Buffer*)` Wenn ein Knoten einen neuen Buffer empfängt, dann wird die `processBuffer()` Methode aufgerufen. Diese Methode sollte wiederum einen neuen bzw. verarbeiteten Buffer zurückliefern. Anstatt einen Buffer zurückzugeben, kann auch ein Event zurückgegeben werden, das an alle Nachfolgerknoten in-stream weitergeleitet wird. Wenn ein eingehender Buffer nicht sofort weiterverarbeitet werden kann, weil z.B. mehrere Buffer erstellt wurden, die zuerst weggeschickt werden müssen, dann kann ein “working flag” gesetzt werden. In diesem Fall wird

die `processBuffer()` Methode ohne neuen Buffer wieder aufgerufen. Um das Flag zu setzen bzw. zu löschen, kann die `setWorkingFlag(bool)` Methode benutzt werden.

In einem Quellknoten kann ein "producing-flag" gesetzt werden. Solange dieses Flag gesetzt ist, wird seine `proccessBuffer()`-Methode aufgerufen, ansonsten wird diese Methode nicht mehr aufgerufen. Beispielsweise kann so sichergestellt werden, dass keine Buffer mehr verschickt werden, wenn das Ende einer Datei erreicht wird, die vom Knoten gelesen und verschickt wird. Die `setProducingFlag(bool)`-Methode kann aufgerufen werden, um das Flag zu setzen bzw. zu löschen.

Ein Quellknoten empfängt keine Buffer, seine `processBuffer()` Methode wird ständig ohne Buffer aufgerufen. In dieser Methode sollte der Knoten neue Buffer generieren, indem er Daten von einer Library, Datei oder Gerät, ... liest.

`Result doStop()` Wenn die `stop()` Methode aufgerufen wird, beendet der Knoten die Verarbeitung und Weiterleitung von Buffern. Ähnlich wie bei der `doStart()` Methode, braucht ein Programmierer die `doStop()` Methode meist nicht zu implementieren.

`Result doFlush()` Innerhalb der `doFlush()` Methode sollte der Knoten, abhängig von seiner Funktionalität alle internen Buffer löschen.

`Result doDeactivate()` In dieser Methode sollte der Knoten alle belegten Ressourcen wieder freigeben, die in der `doActivate()` Methode angefordert wurden.

`Result doDeinit()` Der Knoten sollte alle Ressourcen freigeben, die in der `doInit()` Methode belegt wurden.

Tabelle 3.1 zeigt ein Beispiel für einige Template-Methoden der Knoten im MP3 Player Beispiel.

3.5.1 Beispiel

Als Beispiel für eine einfache NMM Anwendung soll eine Datei von Festplatte gelesen werden, alle vorkommen eines bestimmten Zeichens durch ein anderes ersetzt werden und in eine neue Datei zurückgeschrieben werden. Dabei wird auf zwei vorhandene Knoten zurückgegriffen: Der GenericReadNode liest eine Datei und verschickt deren Inhalt als Buffer. Der GenericWriteNode schreibt empfangende Daten in eine Datei. Das Event *filename* kann an beide Knoten geschickt werden, um die zu lesende bzw. zu schreibende Datei zu bestimmen. Der GenericReadNode schickt das Event *end_track*, wenn die Datei komplett gelesen wurde. Der GenericWriteNode schließt beim Empfang dieses Events die zu schreibende Datei.

Es soll nun ein Knoten entwickelt werden, der alle vorkommen eines vorher definierten Zeichens in einem Buffer durch ein anderes ersetzt.

Entwicklung eines Knotens

Der zu entwickelnde Knoten soll den Namen “SubstituteNode” tragen. Er hat einen Eingang und einen Ausgang und wird von der Klasse `GenericProcessorNode` abgeleitet (vgl. Abschnitt 3.5). Weiterhin soll der Knoten auf das Event *setSubstitution* reagieren, mit dem das zu ersetzende Zeichen bestimmt werden kann. Die zugehörige Handler-Funktion wird `eventSetSubstitution` genannt:

```
class SubstituteNode: public GenericProcessorNode {
  public:
  SubstituteNode( const char* = "SubstituteNode",
              StreamQueue::Mode = StreamQueue::MODE_SUSPEND,
              int = 20 );

  /**
   * Die Verarbeitungsmethode, ihr wird der
   * empfangende Buffer übergeben.
   * Zurückgegeben wird ein neuer Buffer
   * oder ein Event
   */
  Message* processBuffer( Buffer* in_buffer );

  /**
   * Bestimmt das Zeichen c, das durch
   * das Zeichen s ersetzt werden soll
   */
  Result eventSetSubstitution(char& c, char& s);

  private:
  /* Zeichen, das ersetzt werden soll */
  char char_to_replace:

  /* Ersetzungszeichen */
  char replacement;
}
```

Im Konstruktor wird das Ein- und Ausgangsformt und das zu ersetzende Zeichen angegeben, hier wird auch die Handler-Methode für das Event registriert:

```
SubstituteNode::SubstituteNode {
  /* Das Ein- und Ausgangsformat ist ein normaler ASCII Text */
  getOwnInputProperty()->addNewFormat("text/ascii");
  getOwnOutputProperty()->addNewFormat("text/ascii");

  /* registriere Event-Handler Methode */
  registerSetEvent("setSubstitution",
    new TEDObject2<SubstituteNode, char, char>(
```

```
          this,
          &SubstituteNode::eventSetSubstitution
    )
  );

  /* ersetze standardmäßig das Zeichen x durch y */
  eventSetSubstitution( 'x', 'y' );
}

/* Implementation der Event-Handler Methode */
Result SubstituteNode::eventSetSubstitution( char& c, char&
    s) {
  char_to_replace = c;
  replacement = s;

  return( SUCCESS );
}
```

Als nächstes folgt die Implementation der `processBuffer` Methode. Diese Methode führt die eigentliche Ersetzung der Zeichen durch:

```
Message* SubstituteNode::processBuffer( Buffer* in_buffer ){
  /* Speichere Größe des empfangenen Buffers */
  unsigned int length = in_buffer->getUsed();

  /* Fordere Zeiger auf den Datenbereich des Buffers an */
  char* data     = in_buffer->getData();

  /* ersetzte alle Zeichen innerhalb des Buffers */
  for( unsigned int i=0; i<length; i++ ){
    if ( data[i] == char_to_replace ){
      data[i] = replacement;
    }
  }

  /* verschicke veränderten Buffer */
  return( in_buffer );

}
```

Jetzt kann die Anwendung erstellt werden. Es werden Instanzen der 3 Knoten angelegt und diese über Events konfiguriert. Die Anwendung selbst registriert sich für das *end_track* Event, um festzustellen, wann die Anwendung beendet werden kann.

```
/* Diese Klasse wird benötigt , um das end_track Event
   von den Knoten abzufangen
*/
class Listener {
private:
  bool finished_flag;
```

```
public:
  Listener() {
    finished_flag = false;
  }
  /* Handler-Methode */
  Result finished() {
    finished_flag = true;
    return SUCCESS;
  }
  bool getfinishedflag() {
    return finished_flag;
  }
};

int main() {
  Listener* l = new Listener();

  GenericReadNode* readfile = new GenericReadNode();
  /* Bestimme zu lesende Datei ( schicke Event an Knoten) */
  readfile->sendSetEvent(Event("filename",
    new TInValue<string>("Quelldatei.txt")
  );

  SubstituteNode* substitute = new SubstituteNode();
  /* Bestimme zu ersetzendes Zeichen */
  substitute->sendSetEvent(Event("setSubstitution",
    new TInValue<char,char>('a', 'b')
  );

  GenericWriteNode* writefile = new GenericWriteNode();
  /* Bestimme zu schreibende Datei */
  writefile->sendSetEvent(Event("filename",
    new TInValue<string>("Zieldatei.txt")
  );
  /* Registriere Event-Handler Methode der Anwendung */
  writefile->requestSetEventDispatcher()->
      registerEventListener(
    "end_track", new TEDObject0<Listener>(l, &Listener::
        finished)
  );

  try {
    /* Initialisiere Knoten */
    readfile -> init();
    substitute -> init();
    writefile -> init();

    /* Verbinde Knoten */
    Format* format = new Format();
```

```
    readfile -> connectOutputTo( substitute, format );
    substitute -> connectOutputTo( writefile, format );

    /* Aktiviere Knoten */
    readfile -> activate();
    substitute -> activate();
    writefile -> activate();

    /* Starte Knoten */
    readfile -> start();
    substitute -> start();
    writefile -> start();

  } catch(...) {
    /* Fehlerbehandlung */
    exit(1);
  }

  /* Prüfe, ob die Datei verarbeitet wurde */
  while( !(l->getfinishedflag()) );

  /* Stoppe Knoten */
  readfile -> stop();
  substitute -> stop();
  writefile -> stop();

  /* Deaktiviere Knoten */
  readfile -> deactivate();
  substitute -> deactivate();
  writefile -> deactivate();

  /* Uninitialisiere Knoten */
  readfile -> deinit();
  substitute -> deinit();
  writefile -> deinit();

  /* Lösche Knoten */
  delete readfile;
  delete substitute;
  delete writefile;
}
```

Kapitel 4

Auswahl der Hardware

4.1 Anforderungen an die Hardware

Die Multimedia-Box soll bestimmten Anforderungen gerecht werden. Sie soll herkömmliche Geräte wie CD-Player, DVD-Player und TV-Receiver ersetzen. Sie soll erweiterbar und leicht konfigurierbar sein.

Es gibt spezielle Lösungen, wie etwa TV-Receiver mit integrierter Festplatte, mit dem TV Programme direkt auf Festplatte gespeichert werden können. Solche Systeme sind speziell für eine bestimme Aufgabe konzipiert und können nicht erweitert werden. Ein PC hingegen kann fast beliebig erweitert werden und es existieren viele Erweiterungskarten, wie z.B. eine DVB-Karte, mit der digitale TV-Programme empfangen werden können. Deshalb wird als Hardware-Grundlage für die Multimedia-Box Standard-PC-Komponenten verwendet. Die in [65] vorgestellten Hardware-Komponenten werden auch teilweise für die hier vorgestellte Arbeit verwendet. Wichtig bei der Auswahl der Hardware ist auch, dass es geeignete Linux-Treiber dafür gibt, da die Software der Multimedia-Box auf einem Linux-Betriebssystem laufen soll.

Die Multimedia-Box wird aus Standard-PC-Komponenten aufgebaut (Abbildung 4.1). Sie soll so gestaltet werden, dass sie sich von den Standardgeräten im Wohnzimmer kaum unterscheidet, d.h. sie soll sich durch geeignete Wahl des Gehäuses schön integrieren lassen, andererseits sollten die Laufgeräusche der Festplatte und der Lüfter nicht all zu laut sein. Um die Wohnzimmertauglichkeit zu erfüllen, müssen folgende Punkte beachtet werden:

- Das Gehäuse soll eine ansprechende Form haben.
- Laufgeräusche von Prozessor-, Netzteillüfter und Festplatte sollen so gering wie möglich sein.
- Ein normales Fernsehgerät soll an Stelle eines PC-Monitors benutzt werden können.

Abbildung 4.1: Die Multimedia-Box mit LC Display. Das DVD Laufwerk wird durch die aufklappbare Frontblende versteckt.

- Die Multimedia-Box soll komplett mit einer Fernbedienung gesteuert werden können.
- Zur Anzeige von zusätzlichen Informationen, wird die Box mit einem LC-Display ausgestattet.

4.2 Komponenten

4.2.1 Hardware im Überblick

Die nachfolgende Auflistung gibt einen kurzen Überblick über Hardwarekomponenten, die zur Realisierung der Multimedia-Box benutzt wurden. Abbildung 4.2 zeigt den Systemaufbau.

- Micro ATX Gehäuse ATC-600GX1 [34] mit speziellem Dämmaterial (vgl. Abschnitt 4.3.3)
- Netzteil EG365AX-VE von Enermax [20] mit regelbarem Lüfter (vgl. Abschnitt 4.2.3)
- GA-6IEML Micro-ATX Mainboard von Gigabyte (vgl. Abschnitt 4.2.2)
- 1,2 GHz Pentium III CPU (vgl. Abschnitt 4.2.2)
- 256 MB RAM
- 120 GB Festplatte von Maxtor (vgl. Abschnitt 4.2.7)
- NVidia Geforce2 Grafikkarte mit TV-Ausgang (vgl. Abschnitt 4.2.4)

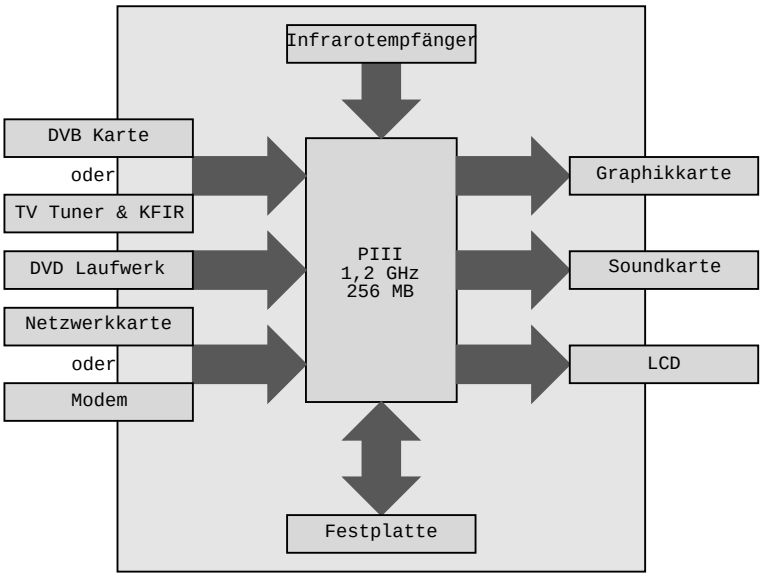

Abbildung 4.2: Möglichkeiten des Hardware-Aufbaus. Als Datenlieferant können DVB-Karte, TV-Tuner-Karte, DVD Laufwerk, Netzwerk oder Modem dienen. Benutzereingaben werden durch einen Infrarotempfänger entgegengenommen. Ausgaben können auf eine Grafikkarte, Soundkarte oder auf ein LC-Display gemacht werden. Die Festplatte dient sowohl als Datenquelle als auch als Ausgabeeinheit.

- Soundblaster Live! 1024 (optional) (vgl. Abschnitt 4.2.5)
- Fujitsu-Siemens DVB-Karte (vgl. Abschnitt 4.2.8) oder TV Tuner und Karte zur MPEG2-Enkodierung (KFIR Karte) (vgl. Abschnitt 4.2.9)
- LG DRD-8160B [47] DVD-ROM Laufwerk (vgl. Abschnitt 4.2.6)
- Infrarotempfänger (vgl. Abschnitt 4.3.1)
- LC-Display (optional) (vgl. Abschnitt 4.3.2)

4.2.2 CPU und Mainboard

Da viele Programme und Libraries für Intel^{TM}CPUs optimiert worden sind, läuft die Multimedia-Box mit einer 1,2 GHz Pentium III CPU. Um digitale Fernsehprogramme oder DVD Filme abzuspielen würde eine 800 MHz CPU reichen, damit die Multimedia-Box aber für zukünftige Erweiterungen genug Performance bietet und verschiedene Funktionen gleichzeitig ausgeführt werden können, wurde sie mit einer schnelleren CPU bestückt.

Als Mainboard wird das GA-6IEML von Gigabyte [28] benutzt. Dieses Micro-ATX Board ist kleiner als die üblichen ATX Boards und passt somit in ein Micro-ATX Gehäuse (Abschnitt 4.3.3). Zur CPU Kühlung wird ein sehr leiser Verax [82] Lüfter benutzt.

4.2.3 Netzwerk

Optional kann die Multimedia-Box auch über eine Internetverbindung verfügen, d.h. sie benötigt eine Netzwerkkarte oder ein Modem. Das in Abschnitt 4.2.2 erwähnte Mainboard verfügt schon über eine On-Board LAN Schnittstelle, so dass kein zusätzlicher PCI-Slot belegt wird. Einigen Softwarekomponenten machen von der Internetverbindung gebrauch, um z.B. über eine Internetdatenbank, die Namen der einzelnen Titel einer Audio-CD abzufragen.

4.2.4 Grafikkarte

Da die Multimedia-Box in der Lage sein soll, Videos über das Fernsehgeräte abzuspielen, muss die Grafikkarte einige besondere Eigenschaften aufweisen:

- Die Auflösung von Videos sollte in Hardware skaliert werden können, da eine Skalierung in Software zu viele CPU-Ressourcen benötigt.
- Unterstützung des YV12 Farbformats, das bei der Video-(De)komprimierung verwendet wird. Damit entfällt die Farbformatkonvertierung nach RGB, die viele CPU-Ressourcen benötigt.
- Bildschirmfüllende TV-Ausgabe.

Die Skalierung der Videos wird benötigt, damit Videos, die in einer kleinen Auflösung gespeichert sind, auf ein bildschirmfüllendes Format gestreckt werden können. Andererseits müssen hochauflösende Videos, die nicht auf den Bildschirm passen, in ihrer Bildgröße reduziert werden. Eine weitere Anwendung der Hardwareskalierung ist die Anpassung der Videos an das korrekte Seitenverhältnis. Beispielsweise werden digitale Satellitenkanäle von Kanal zu Kanal in unterschiedlichen Auflösungen gesendet. Einige werden in 720x576, andere in einer Auflösung von 480x576 gesendet. In beiden Fällen muss das Bild in der Breite gestreckt werden, um das übliche 4:3 Seitenverhältnis des Fernsehgerätes zu erhalten.

Videodekoder erzeugen meist Einzelbilder im YV12 Farbformat. Wird dieses Format direkt von der Grafikkarte unterstützt, so entfällt die Konvertierung in ein Farbformat, das von der Grafikkarte unterstützt wird. Solche Konvertierungen beanspruchen meist viel CPU Zeit.

Auf einem Linux-System wird die Skalierung und das YV12 Format von der Xv-Extension, die Teil des XFree86 Systems ist, bereitgestellt [84]. Xv ist eine Erweiterung des X Systems, das Hardwareskalierung und zusätzliche Farbformate, die speziell für die Wiedergabe von Videos gedacht sind, anbietet.

Um die Multimedia-Box an ein Fernsehgerät anzuschließen, muss die Grafikkarte einen TV Ausgang besitzen.

Alle oben genannten Fähigkeiten werden u.a. von den NVidia [64] Grafikkarten mit TV-Out unterstützt. Das Mainboard hat zwar eine OnBoard Grafikkarte, doch fehlt ihr der TV-Ausgang. Deshalb wird die OnBoard Grafikarte abgeschaltet und durch eine NVidia Geforce2 mit TV-Out erweitert. In Anhang A.1 wird speziell auf die Aktivierung und Konfiguration des TV-Ausgangs der NVidia Boards eingegangen.

In [65] wird ausführlich auf weitere Möglichkeiten eingegangen, wie man u.a. auch Grafikkarten ohne TV-Ausgang mit dem Fernsehgerät verbinden kann. Dort wird eine Schaltung vorgestellt, mit der man den VGA-Ausgang der Karte mit dem Scart-Eingang des Fernsehgerätes verbinden kann.

4.2.5 Sound

Die OnBoard-Soundkarte des Gigabyte Boards kann für normalen Stereosound benutzt werden. Die Multimedia-Box soll aber auch an einen Dolby Digital Dekoder angeschlossen werden können. Diese Decoder verfügen meist über einen S/PDIF Digitaleingang. Die OnBoard Soundkarte hat keinen digitalen Tonausgang, so dass sie durch eine Soundblaster Live! 1024 ersetzt wird. Sie bietet die Möglichkeit unkomprimierte Stereodaten und AC3 kodierte Audiodaten direkt an einen externen Verstärker zu schicken. AC3 kodierte Audiodaten können bis zu 6 Kanäle haben und liegen in komprimierter Form vor [78]. In Anhang A.2 wird beschrieben, wie der Digitalausgang der Soundblaster aktiviert und konfiguriert werden kann.

4.2.6 DVD Laufwerk

Um Videos von einer DVD und Audiodaten von einer CD abspielen zu können, ist ein DVD-Laufwerk in der Multimedia-Box integriert. Damit eine DVD abgespielt werden kann, reicht ein Laufwerk mit einfacher Geschwindigkeit vollkommen aus. Die heutigen Laufwerke lesen DVDs mit mehrfacher Geschwindigkeit aus, dadurch werden auch die Laufgeräusche lauter. Das Laufwerk sollte daher die Möglichkeit bieten, die Lesegeschwindigkeit zu reduzieren (vgl. Anhang A.3).

In der Multimedia-Box wird ein LG DVD-ROM DRD-8160B [47] mit schwarzer Frontblende verwendet. Es kann jedoch jedes beliebige DVD-Laufwerk verwendet werden.

4.2.7 Festplatte

Die Multimedia-Box wird mit einer 120 GB Festplatte von Maxtor ausgestattet. Durch die große Kapazität lassen sich viele Videodateien, insbesondere Fernsehaufnahmen archivieren. Vorteil dieser Festplatte sind auch ihre leisen Laufgeräusche. Es kann aber auch eine andere Festplatte verwendet werden.

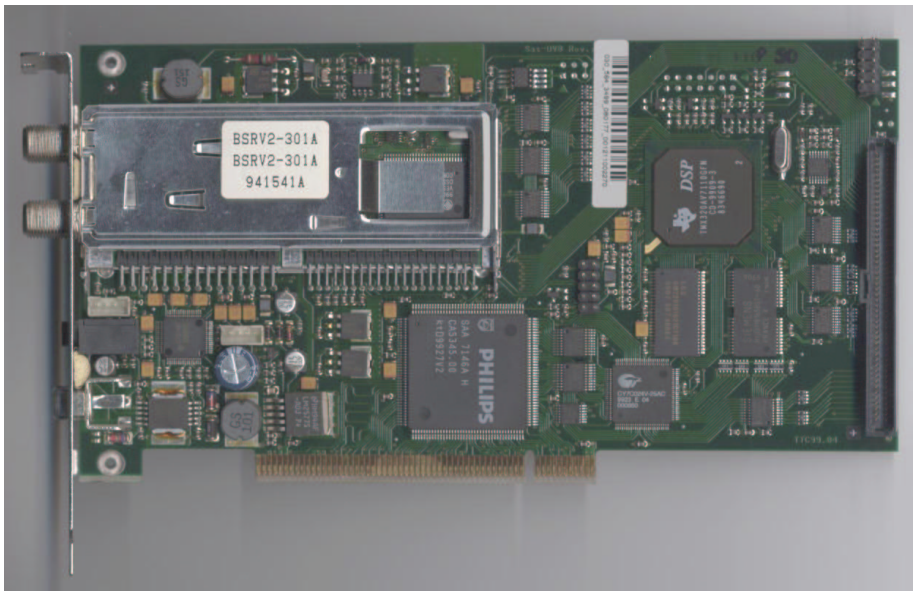

Abbildung 4.3: Die DVB-s Karte, die MPEG2 Audio/Video Daten über Satellit empfangen kann (Bildquelle aus [65]).

4.2.8 DVB Karte

Um digitales Fernsehen über Satellit zu empfangen und aufzunehmen, wird eine DVB-s (Digital Video Broadcast Satellite) Karte (Abbildung 4.3) benötigt. Die Kabelvariante wird DVB-c genannt. Der Datenstrom wird im MPEG2 Format empfangen und kann direkt platzsparend auf einer Festplatte gespeichert werden. In der Multimedia-Box ist eine Fujitsu-Siemens DVB-Karte eingebaut. Sie kann digitale Fernsehprogramme über Satellit empfangen und die komprimierten MPEG2 Video- und Audiodaten mit Hilfe des eingebauten Hardware-Dekoder dekodieren. Die Multimedia-Box benötigt nicht unbedingt den integrierten Hardware-MPEG-Dekoder, da die Software den Datenstrom auch optional ohne Hardware dekodieren kann (Abschnitt 5.2.2.6). Somit kann auch auf weniger teure DVB Karten ohne Hardware-Dekoder wie die Nova-s von Hauppauge [31] zurückgegriffen werden. Installationshinweise zu den DVB-Karten findet man im Anhang A.4.

4.2.9 MPEG2 Enkoderboard

Eine weiter Möglichkeit Videodaten bzw. Fernsehdaten zu empfangen, ist durch einen analogen Satellitenreceiver oder andere analoge Videogeräte. Diese Geräte haben einen analogen Videoausgang in Form eines Composite-Signals oder eines SVHS-Signals. Mit Hilfe eines MPEG Enkoderboards (Abbildung 4.4) können diese analogen Signale in Echtzeit in einen MPEG2 Strom wie er bei DVB üblich ist, gewandelt werden. In Verbindung mit einer TV-Tuner Karte, die terre-

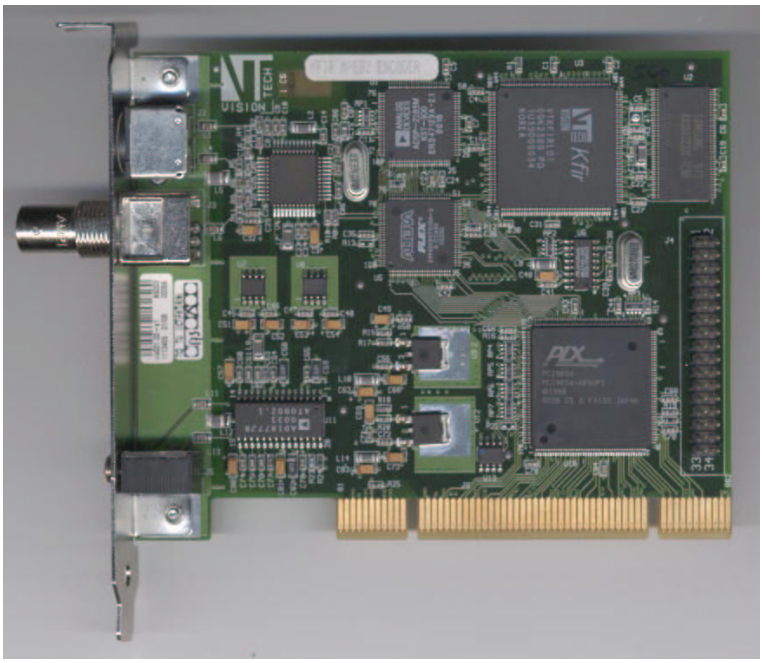

Abbildung 4.4: Das MPEG-Encoderboard mit KFir Chipsatz, das analoge Eingangssignale in einem MPEG2 Videostrom wandelt (Bildquelle aus [65]).

strische TV-Programme empfangen kann, hat man die selben Möglichkeiten wie bei einer DVB Karte und kann somit auch TV-Sendungen platzsparend auf einer Festplatte sichern. In dieser Arbeit wird nur die digitale Variante behandelt (vgl. Abschnitt 4.2.8).

4.2.10 Fernsehgerät

Die Multimedia-Box kann an einen handelsüblichen Fernseher angeschlossen werden, entweder über einen Composite- oder Scart-Eingang.

4.3 Zusätzliche Hardware

Hardwarekomponenten wie Infrarotempfänger und LC-Display können als fertige Module gekauft werden. Insbesondere der Infrarotempfänger und der Anschluss des LC-Displays wurden für die Multimedia-Box aber aus verschiedenen Teilen zusammengebaut. Die Einzelteile sind günstiger als die fertigen Module und lassen sich leicht zusammenbauen.

Abbildung 4.5: Das verwendete LC Display kann 2 Textzeile zu je 40 Zeichen darstellen, angeschlossen wird es an den Parallelport.

Abbildung 4.6: Das Innenleben der Multimedia-Box.

4.3.1 Infrarotempfänger

Eine grundlegende Anforderung an die Multimedia-Box ist, dass sie über eine Fernbedienung gesteuert werden kann. Deshalb benötigt sie einen Infrarotempfänger. Als Basis wird das LIRC (Linux Infrared Remote Control)-Projekt [16] benutzt, in dem die Treiber zur Ansteuerung des Infrarotempfängers entstanden sind.

Der Empfänger ist ausführlich in [15] beschrieben. Es handelt sich um ein kleines Modul, das direkt an die serielle Schnittstelle angeschlossen werden kann. Anhang A.5 zeigt die Stückliste und den Bauplan des Infrarotempfängers wie er auch schon in [65] verwendet wurde.

4.3.2 Liquid Crystal Display (LCD)

Ein LCD (Abbildung 4.5) wird benutzt, um zusätzliche Informationen wie Titel einer CD direkt anzuzeigen.

Das Multimedia-Box LCD ist ein Textdisplay mit 2 Zeilen zu je 40 Zeichen. Es wird mit dem Parallelport verbunden. Gesteuert wird das LCD durch den KS0076B Chip von Samsung Electronics. Der Chip ist bereits komplett mit dem Display auf einem keinen Board montiert. Das Board ist mit einer 16 Pin Buchse bestückt, die verwendet wird, um das LCD mit dem Parallelport zu verbinden. Eine genaue Bauanleitung findet man im Anhang A.6. Das LCD wird mit Hilfe eines speziellen Treibers angesprochen (siehe Anhang A.6).

4.3.3 Gehäuse und Geräuschminimierung

Das Gehäuse trägt die Bezeichnung ATC-600GX1 [34]. Wegen seiner schwarzen Lackierung und der Micro-ATX Bauform, hat es in etwa die Form einer Stereoanlage. Zur Geräuschminimierung wird das spezielle Netzteil EG365AX-VE von Enermax [20] verwendet, dessen Lüfterdrehzahl stufenlos reguliert werden kann. Die Festplatte wird schwingend gelagert, damit die Laufwerksvibrationen nicht auf das Gehäuse übertragen werden. Das ganze Gehäuse ist zusätzlich mit Dämmaterial isoliert. Abbildung 4.6 zeigt das Innenleben der Multimedia-Box.

Kapitel 5

Software Komponenten

Der PC-basierte Ansatz der Multimedia-Box erlaubt ein hohes Maß an Erweiterbarkeit. Dementsprechend muss auch die Software leicht erweiterbar und konfigurierbar sein.

Verschiedene Software-Modulen, müssen implementiert werden, um z.B. CD-Player, MP3-Player, DVD-Player und das Aufnahme und Betrachten von TV-Programmen zu realisieren. In Kapitel 2 wurden einige Software-Implementationen vorgestellt. Nachteil dieser Systeme war, dass sie fast alle monolithisch aufgebaut sind, also nicht viel Freiraum für Erweiterungen bieten.

Die Netzwerk-Integrierte Multimedia Middleware bietet eine modulare und flexible Möglichkeit zur Erstellung von Multimedia-Anwendungen. Die Implementierung verschiedenen Multimedia-Anwendungen wie z.B. ein CD-Spieler, baut deshalb auf NMM auf. Grundlegender Bestandteil von NMM sind die Knoten (vgl. Abschnitt 3.2). Es wird dabei Gebrauch von schon vorhandenen Knoten gemacht [65] oder die Funktionalität ergänzt, indem Knoten erweitert und neue erstellt werden.

Dabei werden häufig schon vorhandene Software-Bibliotheken verwendet, die z.B. komprimierte Audio- oder Videodaten verarbeiten können.

Für die Multimedia-Box sind aber nicht nur die eigentlichen Funktionen grundlegend, sondern auch die Interaktion mit dem Benutzer muss intuitiv und leicht sein. Deshalb muss das System mit einer grafischen Benutzeroberfläche ausgestattet sein, mit der man leicht in Menüs navigieren kann, zwischen Funktionen wechseln kann und die Anwendung konfigurieren kann. Die Benutzereingaben sollen dabei auch mit Hilfe einer Fernbedienung gemacht werden können.

In den folgenden Kapiteln wird die Implementierung einer erweiterbaren grafischen Benutzerschnittstelle vorgestellt. Sie setzt sich aus folgenden Bestandteilen zusammen:

- Grafische Knöpfe, die ausgewählt und selektiert werden können, um z.B. in einem Menü zu navigieren und dort bestimmte Funktionen auszuwählen.
- Fortschrittanzeigen, die den Status eines Vorgangs anzeigen, wie beispielsweise die aktuelle Spielposition eines CD-Titels.

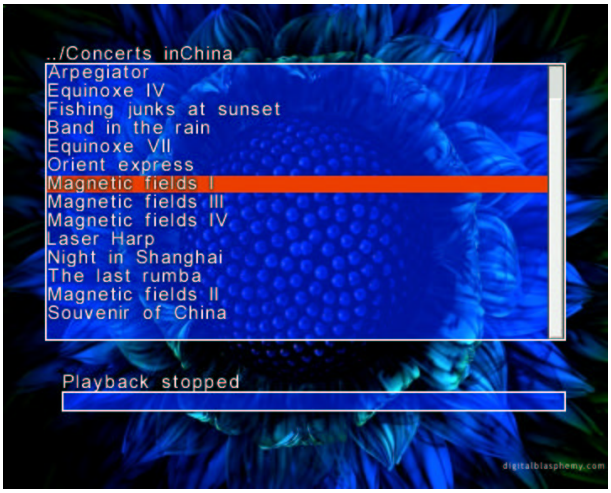

Abbildung 5.1: GUI-Elemente können mit Transparenzen erstellt werden, so dass der Hintergrund durch die Elemente hindurchscheinen kann.

- Listboxen, in denen Texteinträge aufgelistet werden und vom Benutzer selektiert werden können, wie z.B. die Einzelnen Titel einer Audio-CD.
- Ein Mechanismus, mit dem Zeichensätze mit Hilfe einer XML Beschreibung aus einer Grafikdatei ausgelesen werden können.
- Attribute wie Rahmen und verschiedenfarbige Hintergrundebenen, die den grafischen Objekten dynamisch hinzugefügt werden können.

Abschnitt 5.2 befasst sich mit der Entwicklung von NMM Knoten. Dabei werden vorhandene Knoten, die in dieser Arbeit benutzt werden und schon in [65] verwendet wurden, erklärt, einige erweitert und auch Knoten von Grund auf neu entwickelt.

5.1 Grafisches Benutzerinterface

Die Aufgabe eines grafischen Benutzerinterfaces (GUI, Graphical User Interface) ist eine Kommunikation zwischen der Anwendung und dem Benutzer herzustellen. Dabei bedient es sich grafischer Symbole, um dem Benutzer etwas mitzuteilen, bzw. um die Interaktion mit dem Benutzer durchzuführen [41]. Solche grafische Interaktionsobjekte werden *Widgets* genannt. Widgets können einfache Texte sein, Knöpfe, die man per Tastatur oder Maus betätigen kann oder Fortschrittsanzeigen, die den aktuellen Status einer Operation wiederspiegeln [41].

Widgets für Desktop-Anwendungen können beispielsweise mit Hilfe des GUI-Systems QT [80] von Trolltech erstellt werden. Komplexe GUI-Systeme, wie u.a.

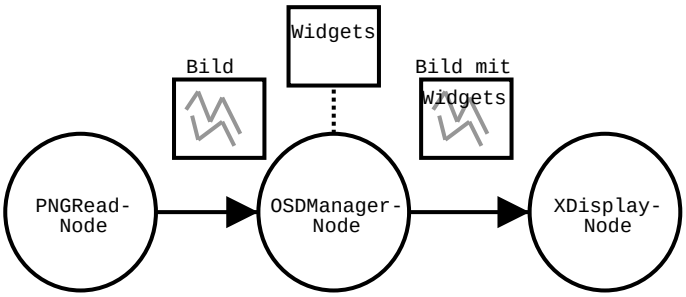

Abbildung 5.2: Schema des OSDManagerNode, der eingehende Videobilder mit grafischen Elementen mischt.

auch das QT-System, bieten viele Möglichkeiten, die grafische Benutzerschnittstelle zu gestalten. Diese GUI-Systeme für Desktop-Anwendungen sind sehr mächtig. Sie bieten ein Fenstersystem, mit dem sich beliebig viele Fenster übereinander legen lassen, verschieben und verkleinern oder vergrößern lassen. Scrollmechanismen um horizontal und vertikal einen Bildausschnitt zu verschieben und vieles mehr.

Desktop GUI-Systeme sind nicht geeignet für eine Settop-Box oder Multimedia-Box. Das Fenstersystem der Multimedia-Box beispielsweise braucht keine Tiefeninformationen auszuwerten, d.h. das Fenstersystem sollte so gestaltet sein, dass ein neues Fenster immer über ein bestehendes Fenster gelegt werden kann, aber nicht darunter oder zwischen zwei andere. Da die Interaktion mit dem Benutzer mit wenigen Tasten einer Fernbedienung durchgeführt werden soll oder anderen “einfachen” Eingabegeräten, wäre die Steuerung zwischen mehreren überlagerten Fenstern für den Benutzer zu unhandlich.

Die GUI-Elemente der Multimedia-Box sollen wie ein On-Screen Display einer herkömmlichen Settop-Box aussehen, d.h. auch, einfach und nicht zu kompliziert. Im Gegensatz zu QT sollen alle grafischen Objekte auch transparent dargestellt werden können, d.h. ein Hintergrundbild oder ein laufendes Video soll durch die grafischen Objekte hindurchscheinen (vgl. Abbildung 5.1).

Widgets müssen auf ein Videobild gezeichnet werden, damit auch bei einer laufenden Anwendung Benutzeraktionen durchgeführt werden können. Der OSDManagerNode ist ein NMM-Knoten, der Widgets auf ein Videobild zeichnen kann (vgl. Abschnitt 5.2.2.1).

Abbildung 5.2 zeigt wie der OSDManagerNode verwendet werden kann. Videobilder, die von einem Leseknoten stammen, werden an den OSDManagerNode weitergeschickt und dort mit verschiedenen GUI-Elementen versehen und danach an einen Displayknoten geschickt, wo das Bild mit allen Widgets schließlich angezeigt wird.

Die Anforderungen an das zu entwickelnde GUI-System lassen sich wie folgt zusammenfassen:

- Widgets müssen auf eine Bitmap gezeichnet werden können, d.h. die Pixeldaten müssen irgendwo im Speicher gehalten werden und abrufbar sein.
- Änderungen wie Textinhalt, Farbe usw., die an einem Widget durchgeführt werden, sollen sich nicht direkt auf das sichbare Objekt auswirken (vgl. Abschnitt 5.1.1).
- Widgets sollen transparente Farben besitzen können.
- die Pixeldaten der Bitmap müssen gegebenenfalls an das Pixelformat des Videobildes angepasst werden (Farbformatkonvertierung).
- Leichte Erstellung und Erweiterung individueller Widgets.
- Hohe Flexibilität, d.h. Widgets sollen mit verschiedenen Attributen versehen werden können, Farbe und Position der Elemente sollen frei wählbar sein.

Die Abbildung 5.1 zeigt ein Beispiel, wie Widgets später in der Multimedia-Box verwendet werden können, um z.B. ein Auswahlfenster, in dem verschiedene Musikdateien aufgelistet sind, darzustellen.

5.1.1 Windowsystem

Damit Widgets auf eine Bitmap gezeichnet werden können, wird als Grundlage für die Darstellung der Widgets ein *Window* benutzt. Das Window speichert das Aussehen der Widgets, d.h. ihre Pixeldaten. Es dient praktisch als Malgrundlage für die Widgets. Windows können in ihrer Größe geändert werden, den Widgets kann also mehr oder weniger Platz gegeben werden, um sich zu zeichnen [41]. Es kann immer nur ein Widget in ein Window gesetzt werden. Das ist aber keine Einschränkung, da mit Hilfe sogenannter Composite-Widgets (siehe Abschnitt 5.1.4) mehrere Widgets zu einem zusammengefasst werden können.

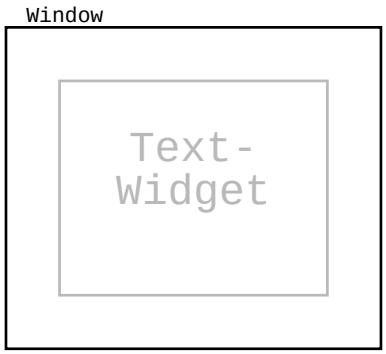

Abbildung 5.3: Ein Window mit Widget

Abbildung 5.3 zeigt ein Window, das ein Widget speichert. In dieser Abbildung werden zwar Rahmen um das Window und das Widget gezeichnet, sie dienen hier aber nur zur Verdeutlichung, wieviel Platz man einem Window zum zeichnen eines Widgets geben kann, bzw. das ein Widget nicht immer ein komplettes Window ausfüllen muss. Widgets werden normalerweise ohne irgendwelche Attribute wie Rahmen und Hintergrundmuster dargestellt, in Abschnitt 5.1.7 wird aber ein System vorgestellt, mit dem man Widgets mit solchen Attributen ausstatten kann.

Je nach Widget lassen sich verschiedene Veränderungen daran vornehmen. So können darin enthaltene Texte verändert oder Farben variiert werden. Würden die Veränderungen unmittelbar das Aussehen des Widgets verändern, so kann es zu einer inkonsistenten Darstellung des Inhalts führen. Angenommen ein Text der durch ein Widget dargestellt werden soll, soll durch einen anderen ersetzt werden. Zuerst wird der alte Text gelöscht, danach der neue Text erstellt. Wird das Widget zu dem Zeitpunkt auf ein Videobild gezeichnet, zu dem der Text gelöscht wurde, so erscheint kurzfristig kein Text mehr. Ein Effekt, den man durch Verwenden eines Windows umgehen kann. Das Window enthält immer ein konsistentes Widget, d.h. Änderungen, die an einem Widget durchgeführt werden, werden erst dann sichtbar, wenn sie in ein Window gezeichnet werden (vgl. Abbildung 5.4).

Window als Pixeldatenspeicher

Windows dienen in erster Linie dazu, Pixeldaten der Widgets zu speichern. Die Klasse `Window` bietet u.a. Methoden zum Erstellen und Löschen des Datenspeichers, im nachfolgenden *Workspace* genannt. Für den Workspace wird eine Struktur angelegt:

```
typedef struct workspace_t {
  unsigned char* data; /* Zeiger auf die Pixeldaten */
  unsigned int width;  /* Breite der Workspace in Pixeln */
  unsigned int height; /* Höhe der Workspace in Pixeln */
  unsigned int stride; /* Anzahl Pixel , um in die nächste Zeile zu
      gelangen*/
}
```

Um ein Window zu erstellen, muss die folgende Methode aufgerufen werden:

```
bool create( int w, int h )
```

Sie erwartet die Breite und Höhe des Workspaces in Pixel und gibt **`true`** zurück, wenn der Workspace erstellt werden konnte. Die Pixeldaten im Workspace werden im RGBA Format gespeichert, d.h. jeder Pixel hat eine Rot-, Grün-, Blau- und Alphakomponente, die jeweils 1 Byte Speicher beanspruchen. Bei einer Workspacegröße von $(Breite * Hoehe)$ werden also $(Breite * Hoehe * 4)$ Bytes Speicher belegt. Die Alphakomponente dient zur Realisierung von Transparenzeffekten, wie in Abschnitt 5.2.2.1 noch genauer beschrieben wird. Bei der Erstellung eines Windows wird der Workspace automatisch mit Nullen gefüllt, d.h. er besteht nur aus transparenten Pixeln.

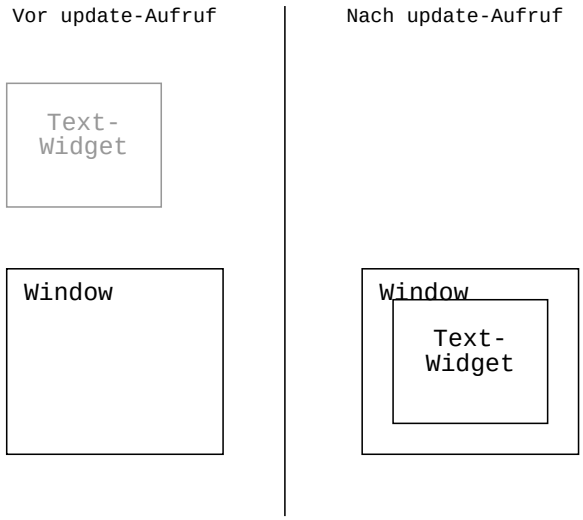

Abbildung 5.4: Das Widget wird erst nach einem update-Aufruf in das Window gezeichnet, es können somit noch Manipulationen am Widget durchgeführt werden ohne dass diese sichtbar werden.

Damit der Workspace mit Daten gefüllt werden kann, muss man zuerst ein Widget hinzufügen. Die verschiedenen Widget-Arten werden noch genau in Abschnitt 5.1.2 erklärt. Das Hinzufügen, genauer gesagt das Setzen eines Widgets, wird mit der Methode

```
void setContent( Widget* vc )
```

der `Window` Klasse durchgeführt. Sie erwartet einen Zeiger auf ein Widget. Mit der `update()` Methode (Abbildung 5.4) erreicht man, dass sich das Widget in das Window zeichnet. Die eigentlichen Pixeldaten des Windows werden mit der Methode

```
unsigned char* get( int* x, int* y, int* w, int *h )
```

abgerufen. Sie liefert einen Zeiger auf die RGBA Pixeldaten zurück, in x und y die Position und in w und h die Breite und Höhe des Windows. Diese Methode wird vom OSDManagerNode aufgerufen, um alle Windows mit ihren Widgets auf ein Videobild zu zeichnen (vgl. Abschnitt 5.2.2.1). Die Position kann mit `void setPosition( int x, int y )` verändert werden und ist später in Abschnitt 5.2.2.1 noch interessant.

Die Methode `hide()` veranlasst, dass die oben erwähnte `get(...)` Methode immer einen NULL-Zeiger zurückliefert. Ein Fenstermanager kann somit prüfen, ob ein Window wirklich gezeichnet werden soll oder nicht. Ein `unhide()` Aufruf macht den Vorgang wieder rückgängig.

Mit der Methode `lock()` kann ein Window gesperrt werden, somit kann kein `update()` ausgeführt werden. Der `update()` Aufruf blockiert so lange bis die

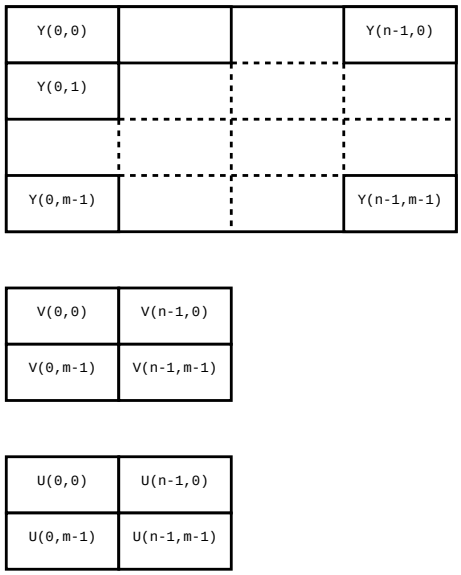

Abbildung 5.5: Aufbau des YV12 Farbformats.

`unlock()` Methode aufgerufen wird. Der OSDManagerNode sperrt das Window, bevor er es auf ein Videobild zeichnet. Somit ist sichergestellt, dass während der Zeichenoperation keine Veränderungen an dessen Inhalt vorgenommen werden können (vgl. Abschnitt 5.2.2.1).

Windows mit anderem Farbformat

In Abschnitt 5.1.1 hat man gesehen, dass eine Workspace aus RGBA Daten besteht. Oft ist es aber sinnvoll, die Pixeldaten in einem anderen Format zu speichern. Vorallem wenn Widgets über einen Videostrom gezeichnet werden sollen. Der Videostrom ist meist im YV12 Farbformat abgelegt [69].

Das YV12 Farbformat wird sehr oft von Videodekodern benutzt. Der Ursprung liegt darin, wie das menschliche Auge seine Umwelt wahrnimmt. Es reagiert viel sensibler auf Helligkeitsunterschiede als auf Farbunterschiede. Das YV12 Farbformat speichert für jeden Pixel die Helligkeitsinformation, den y Wert, aber nur ein Wert pro 2x2 Pixelblock wird für die Speicherung der Farbdifferenzwerte u und v benutzt. Dadurch wird auch eine Komprimierung erreicht.

Die Farbkomponenten des RGB Formats sind im “packed”-Format gespeichert. Das bedeutet, zuerst wird der R, dann der G und dann der B Wert für jedes Pixel gespeichert. Das YV12 Format ist hingegen in “Planes” aufgebaut. Zuerst werden alle y Werte aufgezählt, danach folgt eine Plane für die u und v Werte. Abbildung 5.5 zeigt diesen Aufbau.

Die y, u und v Werte haben einen Wertebereich von -127 bis 128 und können aus den RGB-Werten folgendermassen berechnet werden:

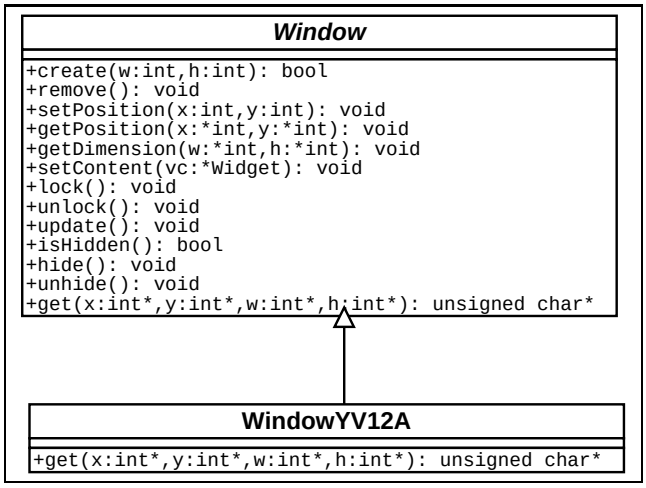

Abbildung 5.6: Klassenhierarchie von `Window`

$$y = 0.299 \cdot r + 0.587 \cdot g + 0.114 \cdot b$$
$$u = (-0.169) \cdot r + (-0.332) \cdot g + 0.500 \cdot b + 128$$
$$v = 0.500 \cdot r + (-0.419) \cdot g + (-0.0813) \cdot b + 128$$

Abgeleitet von der `Window` Klasse ist die `WindowYV12A` Klasse. Sie übernimmt die Konvertierung der RGBA Daten der Widgets nach YV12A. Dieses Format ist aufgebaut wie das YV12 Format, hat jedoch noch eine zusätzliche Alphaplane, damit auch hier die Transparenz der Pixel gespeichert werden können. Die Widgets selbst zeichnen sich weiterhin auf eine RGBA Workspace, nach einem `update()` wird diese Workspace jedoch in eine YV12A Workspace konvertiert. Der Speicherbedarf dieser Workspace beträgt nur $(Breite * Hoehe * 2.5)$ Bytes, da die u und v Planes nur pro 2x2 Pixelblock gespeichert werden. Nach einem Aufruf von

```
unsigned char* get( int* x, int* y, int* w, int *h )
```

erhält man einen Zeiger auf eine YV12A Workspace, die das Widget erhält. Abbildung 5.6 zeigt die Methoden der Windowklassen im Überblick.

5.1.2 Widgets

Die Multimedia-Box soll die Möglichkeit bieten, Menüs darzustellen. Die Menüpunkte sollen durch Grafiken realisiert werden. Menüpunkte, im folgenden auch *Knöpfe* genannt, sollen selektiert und aktiviert werden können.

In einer Auswahlbox, die aus mehreren Textzeilen bestehen kann, sollen verschiedene Informationen abgelegt werden, wie z.B. der Inhalt einer Audio-CD oder eine Verzeichnisstruktur. Innerhalb dieser Box soll der Benutzer die Möglichkeit haben eine Auswahl zu treffen, indem er eine Zeile markieren kann.

Eine Fortschrittanzeige soll dargestellt werden können, um den aktuellen Status einer Operation anzuzeigen, wie z.B. aktuelle Spielzeit einer Audio-CD. Zeitinformationen sollen aber auch genau, d.h. Stunde, Minute und Sekunde angezeigt werden können. Grafiken sollen zur Dekoration der Anwendung beliebig platziert werden können. Je nach GUI-Elemente sollen Form und Farbe variiert werden können.

Der Begriff Widget wurde schon in 5.1 kurz eingeführt. Es handelt sich dabei um grafische Interaktionsobjekte, die man in ein Window setzen kann (siehe Abschnitt 5.1.1), auf einem Display angezeigt werden können und gegebenenfalls auf Eingaben des Benutzers reagieren [41].

Es gibt insgesamt 6 Widgets, im folgenden auch Basis-Widgets genannt, die ausreichen, um ein grafisches User-Interface für die Multimedia-Box mit oben aufgeführten Anforderungen zu implementieren (Abschnitt 5.1.6).

- *ImageView*: ein Widget, um Bilder anzuzeigen. Die Bilddatei muss dabei im PNG-Format [30] vorliegen. Das PNG-Format wurde deshalb ausgewählt, da es u.a. auch Transparenz-Informationen eines Bildes beinhaltet. Die Unterstützung weiterer Formate ist hier denkbar.

- *Buttons*: Widget zum Anzeigen von Knöpfen, die als Bilddatei vorliegen, dabei kann zwischen verschiedenen Knöpfen navigiert werden. Mehrere Knöpfe können dabei aktiviert und deaktiviert werden. Im Augenblick wird dieses Widget von keiner Anwendung verwendet. Es kann jedoch in späteren Anwendungen benutzt werden, um z.B. mehrere *Checkboxen* zu realisieren.

- *RadioButtons*: Ähnlich wie das Button-Widget, hier kann allerdings immer nur ein Knopf aktiv sein, alle anderen Knöpfe werden automatisch deaktiviert. Dieses Widget wird zum Aufbau der Menüs verwendet.

- *ProgressBar*: Anzeige eines Fortschrittbalkens, um den aktuellen Status einer Operation anzuzeigen.

- *TextView*: Wird zur Darstellung von Textzeilen benutzt, dabei kann der Benutzer zwischen Textzeilen navigieren und selektieren.

- *TimeView*: Widget zur Darstellung von Zeitinformationen.

Widgets sollten sich nur auf ihre reine Funktion beschränken. Die Darstellung von zusätzlichen Attributen wie Rahmen, Scrollbalken usw. sollten von anderen Objekten durchgeführt werden (siehe Abschnitt 5.1.7 und Decorator-Design-Pattern [22]).

5.1.3 Schnittstellen der Widget-Klasse

Alle Widgets werden von der Klasse `Widget` (siehe Abbildung 5.7) abgeleitet, diese Klasse definiert die grundlegenden Schnittstellen. Die Navigationsschnittstelle besteht aus fünf Methoden:

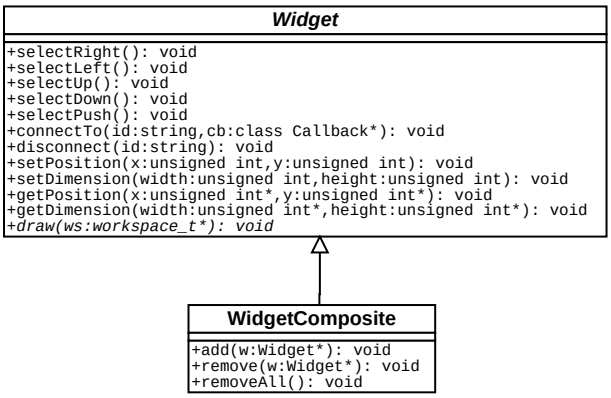

Abbildung 5.7: Das Widget-Interface. Widgets können beliebig innerhalb eines Windows platziert werden und haben eine Navigationsschnittstelle (select-Methoden). Die draw-Methode ist für die Darstellung des Widgets verantwortlich. In einem zusammengesetzten Widget (WidgetComposite) können sich beliebig viele Widgets befinden.

```
virtual void selectRight()
virtual void selectLeft()
virtual void selectUp()
virtual void selectDown()
virtual void selectPush()
```

Wie diese Methoden im einzelnen implementiert sind, ist abhängig von der Funktion des Widgets. Die `selectRight()`, `selectLeft()`, `selectUp()` und `selectDown()` Methoden sollten zur Selektion innerhalb des Widgets benutzt werden, wie z.B. bei dem Button-Widget. Dort werden diese Methoden aufgerufen, um einzelnen Knöpfe anzusteuern, mit `selectPush()` wird der selektierte Knopf aktiviert. Bei ProgressBar-, TimeView- und ImageView-Widget haben die `select`-Methoden keine Funktion. In Abschnitt 6.2.5 sieht man, dass Tastendrücke auf der Infrarotfernbedienung, meist direkt diese Methoden aufrufen.

Callback-Funktionen

Die in Abschnitt 5.1.3 vorgestellte Navigationsschnittstelle bietet die Möglichkeit an, Callback-Funktionen zu registrieren. Man kann beliebig viele Funktionen registrieren, die alle nacheinander aufgerufen werden, wenn eine Navigationsfunktion aufgerufen wird. Eine Callback-Funktion wird durch folgenden Aufruf registriert:

```
void connectTo( string ID, class Callback* cb )
```

Zum einen erwartet die Methode eine ID. Diese ID bezeichnet die entsprechende Navigationsfunktion. Es gibt also insgesamt fünft ID's, die aus einem einfachen String bestehen und nach den Navigationsmethoden benannt sind:

- “selectRight”
- “selectLeft”
- “selectUp”
- “selectDown”
- “selectPush”

Das zweites Argument der `connectTo` Methode ist ein Zeiger auf die Instanz einer Klasse, die von der `Callback` Klasse abgeleitet werden muss, dort muss die **`void`** `run(` **`void`**`*` `)` Methode überschrieben werden:

```
class Callback {
  public:
    virtual ~Callback() {};
    virtual void run( void* ){};
}
```

Eine einfache Vorgehensweise, um eine Callback-Funktion zu registrieren, ist die Benutzung der `TCallback` Klasse. Dabei handelt es sich um ein Template, das einen Klassentyp benötigt, und im Konstruktor einen Zeiger auf die Klasse selbst und die entsprechende Methode erwartet:

```
template<class cl>
class TCallback : public Callback {
  /* die  Callbackfunktion  bekommt einen Zeiger  zurück,
     der  durchgereicht  wird */
  typedef void (cl::*callbackfunc_t) ( void* );

  protected:
    /* Zeiger  auf  die  Klasse  selbst */
    cl*          classdata;

    /* Zeiger  auf  Memberfunktion */
    callbackfunc_t callbackfunc;

  public:
    TCallback( cl* cldata, callbackfunc_t cbfunc ){
      classdata = cldata;
      callbackfunc = cbfunc;
    };

    void run( void* w ){
      ((classdata)->*(callbackfunc))( w );
    };
```

Angenommen eine Klasse `Beispielklasse` hat eine Methode mit dem Namen `callbackfunc1(` **`void`**`*` `)`, dann kann diese Methode so registriert werden:

```
Beispielklasse mycallbacks;
widget->connectTo( "selectLeft",
              new TCallback<Beispielklasse>(
                &mycallbacks,
                &Beispielklasse::callbackfunc1 )
             );
```

Jedesmal, wenn die `selectLeft()` Methode des Widgets aufgerufen wird, wird auch die `callbackfunc1( void* widget )` Methode der `Beispielklasse` aufgerufen, dabei wird ihr ein Zeiger auf das Widget übergeben. Alle registrierten Callback-Funktionen können auf einmal durch folgenden Aufruf unregistriert werden:

```
void disconnect( string id )
```

Abschließend sei noch darauf hingewiesen, dass der obige Mechanismus zum Aufrufen von Callback-Funktionen zum Zeitpunkt entwickelt wurde, als der Event-Mechanismus von NMM noch nicht in der Form existierte wie es heute der Fall ist (vgl. Abschnitt 3.3). Die (zukünftige) Umstellung auf NMM-Events ist hier sicherlich sinnvoll.

Position und Größe

Widgets werden einerseits durch die Größe des Windows, in dem sie platziert sind, beschränkt, können in ihrer Größe aber auch angepasst und im Window bzw. in einem Composite Widget (Abschnitt 5.1.4) verschoben werden. Zum Ändern oder Abfragen der Position stehen folgende Methoden zur Verfügung:

```
void setPosition( unsigned int x, unsigned int y )
void getPosition( unsigned int* x, unsigned int* y )
```

Um die Größe zu Ändern oder Abzufragen benutzt man diese Methoden:

```
void setDimension( unsigned int w, unsigned int h )
void getDimension( unsigned int* w, unsigned int* h )
```

Wird beispielsweise das Window, in dem ein TextView Widget platziert ist, vergrößert, so können pro Textzeile mehr Zeichen und insgesamt mehr Zeilen dargestellt werden.

Darstellung der Widgets

Widgets werden auf Workspaces gezeichnet, die von Windows zur Verfügung gestellt werden. Bekommt ein Window die Anfrage sich neu zu zeichnen, dann wird diese Anfrage an das Widget weitergeleitet (Abbildung 5.8).

Windows rufen dazu die `draw( workspace_t* ws )` Methode des Widgets auf und übergeben ihr dabei einen Workspace, auf die das Widget sich zeichnen kann. Das Farbformat des Workspace ist dabei immer RGBA, eine Farbformatkonvertierung wird gegebenenfalls vom Window selbst durchgeführt (Abschnitt 5.1.1).

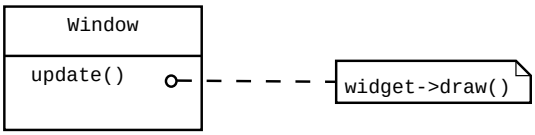

Abbildung 5.8: Soll ein Widget neu gezeichnet werden, bekommt es dies von dem Window mitgeteilt

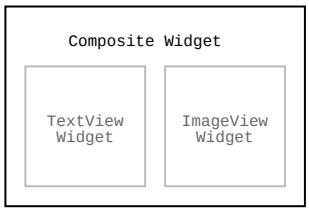

Abbildung 5.9: Composite Widget: Mehrere Widgets werden zu einem Widget zusammengefasst

Die Farbformatkonvertierung wird nur bei einem `update`-Aufruf des Windows durchgeführt und nicht jedesmal, wenn der OSDManagerNode das Window zeichnet. Zusätzlichen Attributen wie Rahmen, Scrollbalken usw. werden von einem Decorator (Abschnitt 5.1.7) gezeichnet. Decorator haben die Aufgabe Windows mit Attributen zu versehen. Sie zeichnen Rahmen um Widgets, versehen sie mit Scrollbalken oder Ändern deren Hintergrund.

5.1.4 Composite Widget

Windows können genau ein Widget aufnehmen, oft ist es aber wünschenswert mehrere Widgets in ein Window zu setzen, um etwa einen Text mit einer Grafik zu versehen. Mit Hilfe des Composite Widgets lassen sich mehrere Widgets zu einem Widget zusammenfassen (Abbildung 5.9). Da ein Composite Widget auch ein normales Widget ist, lassen sich wiederum auch Composite Widgets zu einem Composite Widget hinzufügen. Ein Composite Widget bietet ein Interface an, um Widgets hinzuzufügen oder zu entfernen:

```
void add( Widget* wg)
void remove( Widget* wg )
```

5.1.5 Vorarbeiten zur Entwicklung der Basis-Widgets

Bevor die Basis-Widgets und deren Aufbau beschrieben wird, sind noch einige Vorarbeiten zu leisten. Die in 5.1.6 beschriebenen Basis-Widgets basieren meist auf der Darstellung von Grafiken und Texten. Zur einfachen Verwaltung von Grafiken und Texten werden im folgenden zwei Klassen vorgestellt.

Die Klasse Bitmapreader

Fast jedes Widget greift in irgendeiner Form auf Grafikdateien zu. Button-Widgets brauchen Grafiken zum Darstellen von Knöpfen und auch das TextView-Widget liest seine Zeichensatz-Informationen aus einer Grafikdatei. Sinnvoll ist daher ein Objekt, dass die Verwaltung der Grafiken übernimmt.

Die `Bitmapreader` Klasse bietet Funktionen zum Einlesen, Verwalten und Generieren von Grafikdateien. Unterstützt werden Dateien im PNG-Format [30]. Jeder eingelesenen PNG-Datei kann eine ID zugewiesen werden, um sie später (Abschnitt 5.1.6) referenzieren zu können. Es wird auch die Möglichkeit geboten, Farbformatkonvertierungen durchzuführen, es kann insbesondere zwischen den Formaten RGBA, YV12 und YV12A ausgewählt werden.

Neben dem Einlesen von Bildern, können auch einfache Grafiken generiert werden, wie z.B. einfache Rechtecke, die zur Selektion von Items benutzt werden können.

Eingelesen wird eine PNG-Datei mit:

```
bool loadImage( string id, string filename, format f );
```

Dabei kann der eingelesenen Grafik eine ID zugeordnet werden. Ist die ID schon vorhanden, wird die darunter referenzierte Grafik mit der neuen Datei überschrieben. Neben dem Filenamen der PNG-Datei, kann noch das Farbformat angegeben werden, in dem das Bild im Speicher gehalten werden soll. Als Format kann `RGBA`, `YV12` oder `YV12A` angegeben werden. Die Methode

```
bool createImage( string id, unsigned int width,
                  unsigned int height,
                  unsigned int color1, unsigned int color2,
                  unsigned int color3, format f);
```

erstellt eine Grafik mit der angegeben ID und der angegebenen Breite `width` und Höhe `height`. Die Grafik hat die Form eines Rechtecks. Dabei bestimmt `color1` die Farbe (im RGBA Format) der Grundfläche, `color2` die Farbe der oberen und linken Kante und `color3` die Farbe der rechten und unteren Kante. Bei geeigneter Farbwahl kann so ein 3D Effekt erzeugt werden (Abbildung 5.10). Hier kann

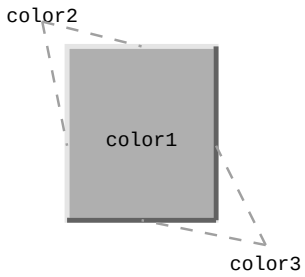

Abbildung 5.10: Der Bitmapreader kann auch einfache 3D-Rechtecke erstellen

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z ¥ ¦ § ¨ © ª « ¬ - ® ¯ ° ± ² ³ ´ µ ¶ ·
1 2 3 4 5 6 7 8 9 0 ' " ! ? % / = ( ) , ; : . - < > # & @ { } [ ] $ * + \ ^ _ ` | ~ ¡ ¢ £ ¤
¸ ¹ º » ¼ ½ ¾ ¿ À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý
Þ ß à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ Œ œ Š š Ÿ Ž ž ƒ ˆ ˜ ‰ € ™

Abbildung 5.11: PNG-Datei, die das Aussehen des Zeichensatzes wiedergibt. Der abgebildete Zeichensatz wurde vom MPlayer übernommen [72].

wiederum das Farbformat festgelegt werden. Eine oder alle Grafiken kann man aus dem Speicher wieder entfernen mit:

```
void freeImage( string id );
void freeAll();
```

An die eigentlichen Bilddaten kommt man durch Aufruf der folgenden Methode heran:

```
unsigned char* getImage( string id,
                         unsigned int* width,
                         unsigned int* height );
```

Es wird ein Zeiger auf die Bilddaten mit der angegebenen ID zurückgegeben, sowie Breite und Höhe der Grafik. Zu beachten ist, dass die Pixeldaten in dem Farbformat zurückgegeben werden, das man beim Laden bzw. Generieren angegeben hat.

BitmapFont

Mit dem in Abschnitt 5.1.5 vorgestellten Bitmapreader ist es auch möglich, Grafiken anzuzeigen, die Wörter oder ganze Texte enthalten. Diese Fähigkeit kann man sich bei statischen Texten zu Nutze machen, um zum Beispiel in einem Hauptmenü, das sich nicht ständig ändert, verschiedene Optionen anzuzeigen.

Soll eine Liste aller MP3-Dateien in einem Verzeichnis angezeigt werden, dann kann auf diese statischen Textbitmaps nicht zurückgegriffen werden. Texte müssen also dynamisch erzeugt werden können. Texte können einfach generiert werden, indem man aus einer Grafikdateien, die die Zeichen des Alphabets und Sonderzeichen enthalten, entsprechende Zeichen extrahiert und aneinanderreiht. Damit die richtigen Bilddaten für das gewünschte Zeichen extrahiert werden können, muss genau definiert sein, welcher Bildbereich der Bitmapgrafik für ein bestimmtes Zeichen steht.

Die Klasse `BitmapFont` bietet die Möglichkeit, aus einer XML-Datei die Beschreibung der Bildbereiche für einzelnen Zeichen auszulesen und abhängig davon ein Interface zur Verfügung zu stellen, um die Bilddaten der einzelnen Zeichen zu verwalten.

Eine XML-Datei, die einen Zeichensatz beschreibt, kann wie folgt aussehen:

```
<FONTS>
```

```
 <FONT name="Arial" type="description" bmapfile="font.png">
  <CH char="A" x="3" y="9" w="18" h="20"/>
  <CH char="B" x="26" y="9" w="16" h="20"/>
  <CH char="C" x="47" y="9" w="17" h="20"/>

  <CH char="." x="1384" y="9" w="7" h="20"/>

  <CH ascii=0 x="1185" y="9" w="13" h="20"/>
 </FONT>
</FONTS>
```

Zwischen den `<FONTS>` Tags können beliebig viele Zeichensatzbeschreibungen stehen, die durch ein `<FONT>` Tag geklammert werden. Attribute dieses Tags sind zu einem `name`, das dem Zeichensatz einen belieben Namen gibt, das `type` Tag, das in der aktuellen Version immer den Wert “description” haben muss und das Tag `bmapfile`, das auf eine PNG-Datei weist, die die Bitmapdaten für den Zeichensatz enthält (vgl. Abbildung 5.11). Da PNG-Dateien u.a. auch Transparenzen speichern, können Zeichensätze benutzt werden, deren Zeichen weiche Kanten besitzen. Zwischen den `FONT` Tags steht für jedes zu beschreibende Zeichen ein `CH` Tag, das mit folgenden Attributen versehen werden kann:

- `char`: Zeichen, das beschrieben werden soll
- `ascii`: ASCII-Nummer des Zeichens, das beschrieben werden soll
- `x`, `y`: Linke obere Ecke des Zeichens in der PNG-Datei
- `w`: Breite des Zeichens in Pixeln
- `h`: Höhe des Zeichens in Pixeln

Im obigen Beispiel wird ein Zeichensatz mit dem Namen “Arial” definiert. Das Aussehen der einzelnen Zeichen ist in der PNG-Datei mit Namen “font.png” gespeichert, die wie in Abbildung 5.11 gezeigt, aussehen kann. Für das Zeichen “A” zum Beispiel wird ab der Position 3,9 eine Grafik mit der Breite 18 und Höhe 19 extrahiert.

Die Zeichensatzverwaltung ist in der Klasse `BitmapFont` implementiert (Abbildung 5.12), die folgende Methoden hat:

Result load(string filename, string fontname) Liest eine XML-Datei ein, die einen oder mehrere Zeichensätze beschreibt und extrahiert die Zeichen des Zeichensatzes mit dem Namen `fontname`. Konnte der Zeichensatz erfolgreich geladen werden, wird `SUCCESS` zurückgeliefert.

void checkDimension(unsigned char c, unsigned int *w, unsigned int *h) Liefert Breite `w` und Höhe `h` des Zeichens `c` zurück.

unsigned char* getPixelData(unsigned char c) Gibt einen Zeiger auf die im RGBA Format gespeicherten Pixeldaten des Zeichens `c` zurück.

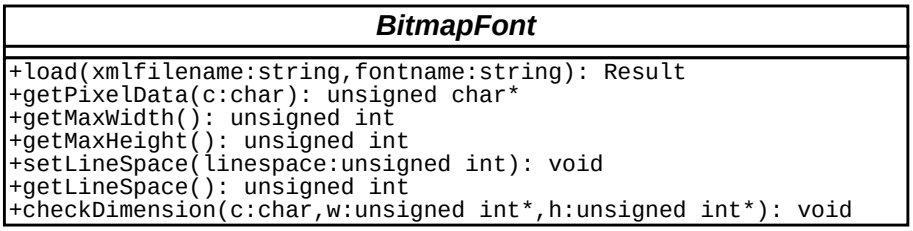

Abbildung 5.12: Interface der BitmapFont Klasse.

unsigned int getMaxWidth() Liefert die maximale Breite eines Zeichens im geladenen Zeichensatz.

unsigned int getMaxHeight() Liefert die maximale Höhe eines Zeichens.

void setLineSpace(unsigned int space) Bestimmt den Abstand zwischen zwei Zeilen.

unsigned int getLineSpace() Gibt den Abstand zwischen zwei Zeilen zurück.

5.1.6 Basis-Widgets

Basis-Widgets bilden die Grundlage für die Interaktion mit dem Benutzer. Sie stellen Texte, Grafiken und Knöpfe dar und reagieren auf Benutzereingaben. Die in Abschnitt 5.1.5 vorgestellten Mechanismen werden u.a. im folgenden benutzt, um die Basis-Widgets zu generieren.

ImageView Widget

Das ImageView Widgets kann Bilder laden und anzeigen. Die Bilddaten werden mit Hilfe eines `Bitmapreader` gespeichert und dem Widget zur Verfügung stellt. Setzen kann man den Bitmapreader mit

```
void setBitmapReader( BitmapReader* br );
```

Der Bitmapreader kann mehrere Grafiken beinhalten. Mit dem Aufruf

```
void setBitmapID( string id );
```

kann eine bestimmte Grafik zum Zeichnen ausgewählt werden, dazu muss die ID der Grafik angegeben werden.

Im Allgemeinen wird immer genau ein Bitmapreader für die Anwendung erstellt. Alle Widgets sollten dann diesen einen Bitmapreader verwenden.

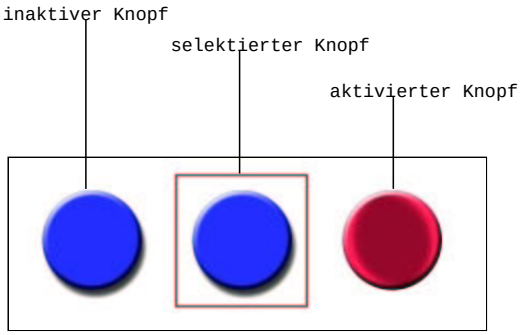

Abbildung 5.13: Das Buttons Widget zeigt Knöpfe an, von denen mehrere aktiviert werden können, aber jeweils nur einer selektiert sein kann. Die Knöpfe müssen als PNG-Dateien vorliegen.

Buttons Widget

Das Buttons Widget wird zum Anzeigen von Knöpfen, die als Bilddatei vorliegen, benutzt, dabei kann zwischen verschiedenen Knöpfen navigiert werden. Mehrere Knöpfe können dabei aktiviert oder deaktiviert werden. Für jeden Knopf gibt es jeweils drei Grafiken. Eine, die den Knopf im deaktivierten Zustand zeigt, eine, die den Knopf im aktivierten Zustand zeigt und eine weitere die den selektierten Zustand wiedergibt. Der selektierte Zustand wird zur Hervorhebung eines Knopfes verwendet, damit der Benutzer sieht, auf welchem Knopf er sich gerade befindet.

Wenn ein Knopf aktiviert oder deaktiviert wird, dann wird die entsprechende Grafik eingeblendet. Eine Besonderheit der selektierten Grafik ist, dass sie über eine aktivierte oder deaktivierte Grafik gezeichnet wird, weiterhin kann immer nur ein Knopf eine Selektion erhalten (Abbildung 5.13). Wird ein Knopf selektiert, dann wird automatisch die Selektion eines vorherigen Knopfes aufgehoben. Im Rahmen dieser Arbeit wird von der aktivierten Grafik kein Gebrauch gemacht. Denkbar ist hier die Darstellung einer Checkbox, in der die aktivierte Grafik die Form eines Kreuzes hat, um eine aktive Checkbox zu symbolisieren.

Jedem Knopf kann eine ID zugeordnet werden und ein Label. Die ID ist eine Zahl, d.h. Knöpfe werden in der Regel durchnummeriert. Im Label kann eine kurze Funktionsbeschreibung des Knopfes stehen, wie z.B. “volume up/down”. Wird der Knopf aktiviert/deaktiviert kann an Hand des Label die Funktion des Knopfes ermittelt werden.

void addButton(int buttonid, string label, int x, int y, string inactive_id, string active_id, string select_id) Hiermit kann ein Knopf hinzufügen werden, dabei muss eine ID für den Knopf vergeben werden, ein Label, die obere, linke Ecke (x/y), an der der Knopf positioniert werden soll, sowie drei IDs von Grafiken des Bitmapreaders, die für die Darstellung (deaktiviert, aktiviert und selektiert) des Knopfes benutzt werden sollen.

void deleteButton(int id) Löscht den Knopf mit der angegebenen ID.

void deleteAllButtons() Löscht alle Knöpfe.

string getButtonLabel(int buttonid) Liefert den Label eines Knopfes.

string getSelectedButtonLabel() Um den Label des gerade selektierten Knopfes zu erhalten kann diese Methode benutzt werden.

bool getButtonState(int buttonid) Diese Funktion liefert `true` zurück, wenn der angegebene Knopf aktiviert ist.

void activateButton(int buttonid) Aktiviert den angegebene Knopf. Hierzu kann auch die in Abschnitt 5.1.3 vorgestellte Methode `selectPush()` benutzt werden.

void deActivateButton(int buttonid) Deaktiviert den Knopf mit der angegebenen ID

void deActivateAll() Deaktiviert alle Knöpfe.

int selectButton(int buttonid) Hiermit wird der Knopf mit der angegebenen ID selektiert. Auch mit den Navigationsmethoden kann ein Knopf selektiert werden. Dabei wird mit `selectLeft()` die ID des zu selektierenden Knopfes um eins erniedrigt und mit `selectRight()` um eins erhöht. Soll die ID um mehr als eins erhöht oder erniedrigt werden, benutzt man `selectUp()` und `selectDown()`.

void setStep(unsigned int s) Der Erhöhungsschritt für die oben genannten `select` Methoden kann hiermit eingestellt werden.

void deSelectButton() Hebt die Selektion eines Knopfes auf.

RadioButtons Widget

RadioButtons haben dieselben Eigenschaften wie Buttons Widgets. Der Unterschied liegt in der Behandlung der aktivierten Knöpfe. Wird ein Knopf aktiviert, dann werden automatisch alle anderen Knöpfe deaktiviert. Es ist also maximal ein Knopf aktiv.

ProgressBar Widget

Mit einem ProgressBar Widget kann ein Fortschrittbalken erstellt werden, um zum Beispiel bei einem MP3-Player den abgespielten Teil eines Liedes anzuzeigen. Ein Fortschrittbalken hat eine definierte Maximallänge, wie die Länge eines MP3-Files. Die Länge des Balkens beschreibt den aktuellen Fortschritt, d.h. bei einem MP3-Player die aktuelle Position im Lied. Der Fortschrittbalken kann mit einer Farbe gefüllt werden und mit sogenannten Spacern, die periodisch als Dekoration

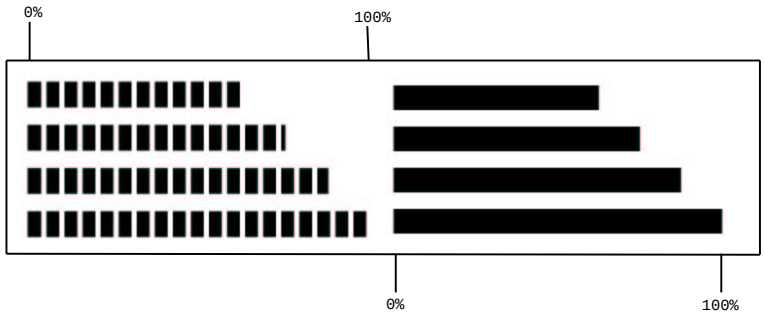

Abbildung 5.14: Verschiedene ProgressBar Widgets mit unterschiedlichen Längen. Fortschrittsanzeigen können mit oder ohne Trennbalken (Spacern) angezeigt werden.

in den Balken eingefügt werden (Abbildung 5.14). Um die Maximallänge eines Fortschrittbalkens zu setzen, wird die Funktion

```
void setMaxProgress( unsigned int max )
```

benutzt. Den aktuellen Fortschritt setzt man mit

```
void setProgress( unsigned int progress )
```

Sinnvoll sind dabei nur Werte, die kleiner als der Maximalwert sind. Die Farbe des Balkens wird gesetzt mit:

```
void setBarColor( unsigned int color )
```

Als Dekoration können periodisch Lücken (Spacer) in den Balken eingefügt werden:

```
void setBarSpaces( unsigned int space,
                   unsigned int width )
```

Dabei ist `space` die Breite der Lücke und `width` die Breite eines Segmentes.

Textview Widget

Das TextView Widget (Abbildung 5.15) wird für die Anzeige von Textzeilen benutzt, dabei kann der Benutzer zwischen Textzeilen navigieren und auswählen. Die Textzeilen werden in einer Liste aufgeführt, die auch über die Größe des Windows hinausgehen kann. Das TextView Widget kann mit einem ScrollDecorator ausgestattet werden, d.h. es besitzt einen Scrollbalken, der das Verhältnis von den gesamten im TextView befindlichen Textzeilen und den gerade angezeigten Textzeilen wiederspiegelt (siehe Abschnitt 5.1.7).

Eine Textzeile kann mehrere Daten speichern. Zum einen, den reinen Text, der zur Ausgabe benutzt wird, zum anderen einen speziellen Text, der als URL

Abbildung 5.15: Ein TextView Widget, das mehrere Textzeilen anzeigt.

Abbildung 5.16: Dasselbe TextView Widget mit geändertem Zeilenabstand

bezeichnet wird und verwendet wird, um den Inhalt der Zeile genauer zu beschreiben. So könnte bei einer Playlist einerseits der Name eines CD-Titels oder MP3-Files angezeigt werden und andererseits kann in der URL der genaue Medientyp bestimmt werden, zum Beispiel “cd:track01” oder “file:musik.mp3” (vgl. Abschnitt 6.3.7). Weitere Daten können in einem “common” Bereich abgelegt werden. Dort kann ein Zeiger auf beliebige Daten hinterlegt werden. Anzeigetext, URL und “common” Zeiger werden in einer Struktur abgelegt:

```
typedef struct {
  string text;
  string url;
  void* common;
} content_t;
```

int add(content_t) Hiermit wird eine Zeile ans Ende der Liste angehängt. Übergabeparameter ist eine `content_t` Struktur. Zurückgeliefert wird die Zeilennummer, in der der Eintrag steht.

int addLine(string text, string url, void* common) Text, URL und “common” Zeiger können mit dieser Methode direkt hinzugefügt werden.

void insert(int pos, content_t) Will man die Zeile nicht ans Ende anfügen, so kann hiermit die Zeilenposition `pos` angegeben werden, an der die Zeile hinzugefügt werden soll.

void insertLine(int pos, string text, string url, void* common) Analog zu `addLine`, jedoch kann hier eine Zeilennummer angegeben werden.

content_t& getLine(unsigned int line) Liest den Inhalt einer Zeile aus, dabei muss die Zeilennummer `line` angegeben werden.

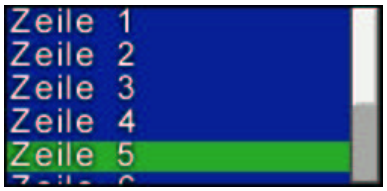

Abbildung 5.17: Ein TextView Widget mit selektierter Zeile. Das Window ist zu klein, um alle Zeilen anzuzeigen, es werden deshalb Zeilen abgeschnitten. Bei aktiviertem “Autofocus” ist die selektierte Zeile immer im sichtbaren Bereich.

string getLineContent(unsigned int line) Gibt den Textanteil der angegebenen Zeile zurück.

string getLineUrl(unsigned int line) Liefert den URL-Eintrag der Zeile.

void* getLineCommon(unsigned int line) Gibt einen Zeiger auf den “common” Eintrag zurück.

void deleteLine(int pos) Löscht die angegebene Zeile.

void deleteAllLines() Löscht alle Zeilen.

void setFont(BitmapFont* font) Setzt den Zeichensatz, mit dem die Textzeilen dargestellt werden.

unsigned int selectLine(unsigned int line) Selektiert die angegebene Zeile (Abbildung 5.17).

unsigned int selectLine2(unsigned int line) Hebt eine (zweite) Zeile hervor.

void deSelectLine() Deselektiert eine Zeile.

void deSelectLine2() Deselektiert die Zeile mit der 2. Hervorhebung.

unsigned int getSelectedLine() Liefert die aktuell selektierte Zeile.

unsigned int getSelectedLine2() Liefert die Zeilennummer, der 2. Hervorhebung.

void focusSelection() Durch Aufruf dieser Methode wird automatisch der Teil der Liste in den sichbaren Bereich gebracht, in dem sich die selektierte Zeile befindet.

void setAutoFocus(bool af) Wird der “Autofocus” aktiviert, so wird automatisch die selektierte Zeile in den sichtbaren Bereich gebracht (Abbildung 5.17).

void setSelectionColor(unsigned int color) Die Farbe des Selektionsbalken kann hiermit gesetzt werden, dabei ist `color` ein RGBA-Wert. item**void setSelectionColor2(unsigned int color)** Bestimmt die Farbe der 2. Hervorhebung.

unsigned int getNumLines() Ermittelt die Anzahl der gesamten Textzeilen.

void setLineSpace(unsigned int ls) Setzt den Zeilenabstand (Abbildung 5.16).

unsigned int getLineSpace() Liefert den Zeilenabstand.

TimeView Widget

Das TimeView Widget (Abbildung 5.18) zeigt Stunden und Sekundeninformationen an. So können bei einem CD-Player die abgespielte oder noch zu spielende Zeit eines Liedes angezeigt werden.

Das Setzen des Zeichensatzes, mit dem die Zeit angezeigt werden soll, wird durch folgende Methode realisiert:

```
void setFont( BitmapFont* bfont );
```

Vor die eigentliche Zeitinformation kann ein Text gesetzt werden:

```
void setPrefix( string p );
```

Das Setzen der Zeit geschieht mit:

```
void setCurrentTime( long t );
```

Um die Zeit abzufragen, muss die nachfolgende Funktion aufgerufen werden:

```
long getCurrentTime();
```

5.1.7 Decorator

Decorators werden verwendet, um Widgets mit zusätzlichen Attributen wie Rahmen, Scrollbalken etc. zu versehen. Eine Möglichkeit, um Widgets mit solchen Attribute zu versehen, ist, neue abgeleitete Klassen zu bilden. Man könnte also eine Klasse erstellen, die von dem entsprechenden Widget abgeleitet ist und die `draw()` Methode überschreiben. Diese Vorgehensweise ist jedoch inflexiebel, da die Auswahl der Attribute statisch ist, d.h. wenn eine abgeleitete Klassen dem

Abbildung 5.18: Ein TimeView Widget, das in dieser Abbildung die aktuelle Uhrzeit wiedergibt.

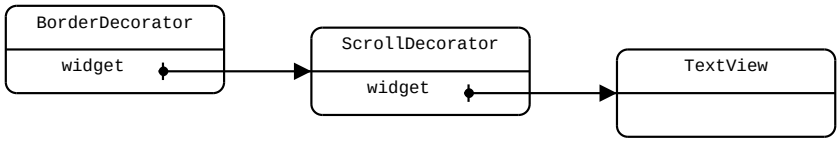

Abbildung 5.19: UML Darstellung des Zusammenhangs von TextView Widget, Scroll- und BorderDecorator

Widget verschiedene Attribute hinzufügt, dann werden die Attribute immer mitgezeichnet ohne dass dies zur Laufzeit entschieden werden kann. Man kann also nicht entscheiden ob und wann man ein Attribut hinzufügen will.

Eine flexiblere Methode bietet das Decorator-Design Pattern, das in [22] beschrieben ist. Die Komponente, die das Attribut zeichnet, wird in ein eigenes Objekt verpackt. Dieses Objekt wird als Decorator bezeichnet. Der Decorator hat das gleiche Interface wie das Widget, das er mit einem Attribut versieht, er wird auch von der `Widget` Klasse abgeleitet. Somit ist die Benutzung von Decoratorn transparent, man kann nicht unterscheiden, ob es sich um ein "reines" Widget oder ein mit Attributen versehenes Widget handelt. Soll ein Widget mit Attributen gezeichnet werden, so leitet der Decorator die Anfrage weiter und kann davor oder danach seine eigenen Zeichenoperationen am Objekt durchführen. Decorator können beliebig hinzugefügt und ausgetauscht werden.

Ein weiterer Vorteil von Decoratorn ist, dass man sie beliebig hintereinanderschalten kann, würde man verschiedene Unterklassen dafür bilden, um alle Kombination der Attribute zu erreichen, würde die Anzahl der Klassen sehr schnell in die Höhe steigen.

Angenommen, man hat ein TextView Widget, welches mehrere Textzeilen in einem Window anzeigt, so dass nicht alle Zeilen auf einmal dargestellt werden können. TextView Widgets haben standardmäßig keine Scrollbalken. Um dynamisch einen Scrollbalken anzuzeigen, kann ein Scroll Decorator hinzugefügt werden. Der Scroll Decorator fordert sich von dem TextView Widget Informationen über Anzahl der Textzeilen, sichtbare Textzeilen und die Position des Selektionsbalken an und zeichnet abhängig davon einen Scrollbalken (vgl. Abbildung 5.17). Decorators können nach Belieben zusammengestellt werden, so dass das Hinzufügen eines Rahmens mit einem Borderdecorator leicht realisierbar ist. Abbildung 5.20 zeigt ein TextView Widget, das mit zwei Decoratoren versehen ist.

Alle Decorator sind Unterklassen der Klasse `Decorator`, die wiederum von der `Widget` Klasse abgeleitet ist. Der Konstruktor erwartet immer einen Zeiger auf ein Widget, dem der Decorator Attribute hinzufügen soll. Attribute werden letztendlich in der `draw()` Methode hinzugefügt. Der Decorator zeichnet sich auf einen Workspace und reicht diesen an das nächste Widget oder Decorator weiter (Abbildung 5.19). Im folgenden werden verschiedene Decorators und deren Interface vorgestellt.

BorderDecorator

Ein Borderdecorator (Abbildung 5.20) zeichnet einen Rahmen um ein Widget, dabei können Farbe und Breite des Rahmens bestimmt werden:

void setBorderWidth(unsigned int bw) Bestimmt die Breite `bw` (in Pixeln) des Rahmens.

void setBorderColor(unsigned int color) Bestimmt die Farbe des Rahmens.

PlaneDecorator

Widgets haben meist einen komplett transparenten Hintergrund, um den Hintergrund mit einer Farbe zu versehen, benutzt man den PlaneDecorator. Er zeichnet hinter einem Widget eine Ebene, deren Farbe man mit

```
void setPlaneColor( unsigned int color );
```

setzen kann.

HeaderDecorator

Oft ist es nützlich, dass man ein Widget mit einer Überschrift versieht. Diese Aufgabe übernimmt der HeaderDecorator, der mit Hilfe eines Bitmapfonts eine Textzeile über das Widget platziert.

void setFont(BitmapFont* bfont) Den Zeichensatz muss man schon im Konstruktor des Decorators angeben, kann aber nachträglich mit dieser Methode verändert werden.

void setText(string str) Bestimmt die Textzeile, die über dem Widget platziert wird.

ScrollDecorator

In den vorherigen Abschnitten wurde die wesentliche Aufgabe des ScrollDecorators schon erwähnt, also das Zeichnen eines Scrollbalkens um ein Widget. Dabei

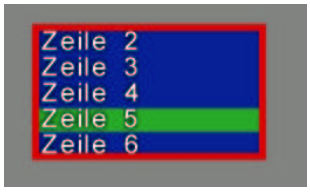

Abbildung 5.20: Ein TextWidget mit Plane- und BorderDecorator.

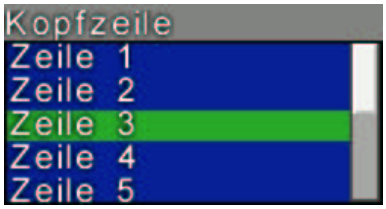

Abbildung 5.21: Ein TextWidget mit Plane-, Border-, Scroll- und HeaderDecorator.

macht der Scrolldecorator nur Sinn, wenn das Widget auch für einen solchen Decorator ausgelegt ist, wie bei einem TextView Widget. Bei einem ProgressBar oder ImageView Widget kann zwar auch ein ScrollbarDecorator hinzugefügt werden, doch die Unterstützung eines Scrollbalkens ist dort nicht implementiert, da diese Funktionalität (bisher) nicht benötigt wird. Denkbar ist hier, dass man mit Hilfe eines ScrollDecorator bei einem ImageView Widget ein Bildausschnitt wählen kann, wenn das anzuzeigende Bild zu groß für eine komplette Darstellung ist.

Das Aussehen des Scrollbalken kann durch einige Methoden variiert werden:

void setBarWidth(unsigned int width) Setzt die Breite `width` (in Pixeln) des Scrollbalkens

void setBackgroundColor(unsigned int bcolor) Bestimmt die Hintergrundfarbe der Scollleiste, `bcolor` gibt auch hier wieder die Farbe im RGBA Format an. Die Hintergrundfarbe bezeichnet die Farbe, die sichtbar wird, wenn der Scrollbalken nicht die ganze Leiste überdeckt.

void setBorderColor(unsigned int bcolor) Um die eigentliche Scrollbarleiste kann ein Rahmen platziert werden, dessen Rahmenfarbe man hiermit bestimmen kann.

void setBarColor(unsigned int bcolor) Setzt die eigentliche Farbe des Scollbalkens

void setBarBorderColors(unsigned int bcolor1, unsigned int bcolor2) Um einen 3D Effekt zu erzielen (siehe auch Abschnitt 5.1.5) kann der Scrollbalken mit einem Rahmen versehen werden, dabei ist `bcolor1` die Farbe der oberen und linken Kante und `bcolor2` die Farbe der rechten und unteren Kante.

5.1.8 Widgetunits

In den vorangegangenen Abschnitten wurden Komponenten vorgestellt, die zur Entwicklung eines Benutzerinterfaces ausreichen. Die Trennung zwischen Pixeldaten (Windows), Interaktionsobjekten (Widgets) und zusätzlichen Attributen (Decorators) lässt zwar ein hohes Maß an Flexibilität zu, doch müssen hier mehrere

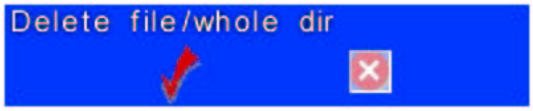

Abbildung 5.22: Eine MessageUnit, die auf eine Bestätigung des Benutzers wartet.

Objekte angelegt werden, um eine ansprechende GUI zu entwickeln. Um komplexere GUI Element leichter zu erstellen, werden sogenannte Widgetunits eingesetzt. Widgetunits fassen Windows, Widgets und Decorators zusammen (vgl. Fassaden-Design-Pattern [22]).

Zum Beispiel kann man mit einer MessageUnit ganz leicht eine Box erstellen, die einen Rahmen, eine Hintergrundebene und einen Text enthält, ohne explizit ein Window, Widgets und die entsprechenden Decorator zu erstellen. Alle Units sind von der Klasse `WidgetUnit` abgeleitet, die zum einen das bekannte Navigationsinterface hat:

```
Result selectLeft()
Result selectRight()
Result selectUp()
Result selectDown()
```

Diese Methoden reichen Anfragen an das entsprechende Widget oder Widgets weiter. WidgetUnits haben eigene Windows, auf die sie sich zeichnen können. Mit

```
Window* getWindow()
```

wird ein Zeiger auf dieses Window zurückgegeben.

MessageUnit

Mit Hilfe einer MessageUnit kann eine Box gezeichnet werden, die einen Hinweistext enthält (siehe Abbildung 5.22). Zusätzlich können drei Knöpfe angegeben werden, zwischen denen der Benutzer navigieren kann. Angenommen man implementiert eine Auswahlbox, in der das aktuelle Directory der Festplatte angezeigt wird, durch drücken der “delete” Taste soll nun die selektierte Datei gelöscht werden. Das Löschen soll nicht unmittelbar nach dem Tastendruck erfolgen, sondern es soll zuerst nachgefragt werden, ob die Datei wirklich gelöscht werden soll. Mit einer MessageUnit kann man zwei Knöpfe mit der Beschriftung “JA” und “NEIN” anlegen, die für solche Abfragen benutzt werden können. Zur Darstellung der Knöpfe wird wieder ein Bitmapreader verwendet. Die Textausgabe wird mit einem Bitmapfont realisiert:

MessageUnit(BitmapReader* br, BitmapFont* font) Im Konstruktor muss ein Bitmapreader angegeben werden, der die Grafiken für die Knöpfe bereitstellt. Die IDs der Grafiken sind für die MessageUnit fest vorgegeben:

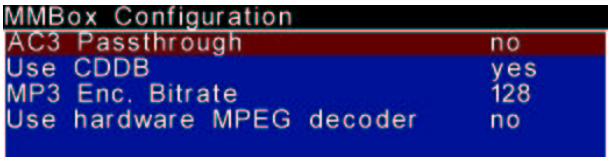

Abbildung 5.23: Eine DoubleListUnit mit HeaderDecorator und PlaneDecorator. Sie ist in zwei Spalten aufgeteilt, links stehen Schlüsselparameter, dessen Werte man rechts ablesen und verändern kann.

`OSD_OK`, `OSD_CANCEL` und `OSD_NO` werden für Knöpfe benutzt die eine Bestätigung, einen Abbruch oder eine Verneinung darstellen. Der Text wird mit einem Bitmapfont dargestellt.

Result print(string headerstr, string text, string label1, string label2, string label3, unsigned int sel) Diese Methode gibt die eigentliche Nachricht, die in `text` steht wieder. Das Argument `headerstr` enthält einen Informationstext, der als Überschrift für das Widget verwendet wird. `label1` enthält die Nachricht, die beim Drücken des `OSD_OK` Knopfes gesendet wird, `label2` die Nachricht für den `OSD_CANCEL` Knopf und `label3` die Nachricht für den `OSD_NO` Knopf.

string messageID() Liefert den gedrückten Knopf zurück, es wird einer der drei oben genannten Label zurückgegeben.

DoubleListUnit

Eine DoubleListUnit (Abbildung 5.23) ist eine Liste mit zwei Spalten. Dabei bietet eine Zeile der rechten Spalte mehrere Inhalte zur Auswahl. Anwendungsmöglichkeit ist ein Konfigurationsmenü. Hier werden in der linken Spalte die möglichen Optionen aufgelistet und in der rechten Spalte Werte zwischen denen ausgewählt werden kann.

DoubleListUnit(BitmapFont* font)

void setColors(unsigned int win_active_color, unsigned int win_inactive_color, unsigned int sel_active_color, unsigned int sel_active_color2, unsigned int sel_inactive_color) Hiermit wird das Aussehen konfiguriert. Man hat insgesamt drei Farbschema. Wenn das Window inaktiv ist, wird die Hintergrundfarbe auf `win_inactive_color` gesetzt und die Farbe des Selektionsbalken auf `sel_inactive_color`. Bei einem aktiven/selektierten Window wird die Hintergrundfarbe auf `win_active_color` gesetzt und die Farbe des Selektionsbalkens entweder auf `sel_active_color` oder auf `sel_active_color2`, je nach Modus.

void selectWindow(unsigned int mode) Selektiert das Window und setzt somit das Farbschema, `mode` gibt das Farbschema an. Ist `mode`=0, dann wird die Farbe `sel_active_color` für den Selektionsbalken benutzt.

void deSelectWindow() Deaktiviert das Window, das entsprechende Farbschema wird gesetzt.

void setHeader(string h) Setzen der Kopfzeile.

void setFirstColumnWidth(unsigned int w) Gibt die Breite der ersten Spalte (in Pixeln) an.

unsigned int addLine(string option, vector<string> values, string url) Fügt eine Zeile hinzu. Dabei ist `option` der Text, der in der linken Spalte erscheint und `values` ist eine Liste von Werten, die man für die Option auswählen kann, `url` kann zusätzliche Informationen enthalten.

string getLineUrl(unsigned int l) Liefert zusätzliche Informationen, die in der angegebenen Zeile `l` gespeichert sind.

string getLineContent(unsigned int l) Liefert den Wert der rechten Spalte der Zeile `l`.

unsigned int getNumLines() Gibt die Anzahl Zeilen der Liste zurück.

void chooseLineContent(unsigned int a, unsigned int b) Setzt den Wert der Zeile `a` auf die Option mit der Nummer `b`.

void chooseLineContent(unsigned int a, string b) Setzt den Wert der Zeile `a` auf die Option mit dem String `b`.

unsigned int getSelectedLine() Liefert die Zeilennummer der selektieren Zeile

void changeSelectedLineContent(string b) Ändert den Text in der rechten Spalte der selektierten Zeile auf `b`.

void changeLineContent(unsigned int l, string b) Ändert den Text in der rechten Spalte der Zeile `l` auf `b`.

void deleteLine(unsigned int pos) Löscht die Zeile mit der Nummer `pos`.

void deleteAllLines() Löscht alle Zeilen.

Weitere Units

Es gibt weitere Units wie ProgressBarUnit, TimeViewUnit und ListUnit, die im Grunde die Funktionalität der entsprechenden Widgets haben, aber mit einem Window und Decorator versehen sind.

5.2 Knoten

Ein wichtiges Ziel ist die leichte Erweiterbarkeit der Multimedia-Box. Wichtige Voraussetzung ist deshalb, dass sie modular aufgebaut ist. Die einzelne Funktionen wie MP3 Dateien Abspielen, CD Abspielen, DVD Abspielen, TV Programme Empfangen und Aufnehmen sind daher nicht in einem einzigen Modul untergebracht. Sie bestehen aus verschiedenen Komponenten oder Knoten, wie sie in NMM genannt werden. Diese Knoten können zu einem Flussgraph verbunden werden und bilden die Grundlage für die eigentliche Multimedia-Box Anwendung (vgl. Kapitel 6).

Im Rahmen dieser Arbeit wurden neue Knoten entwickelt, bestehende Knoten erweitert, aber auch bereits bestehende Knoten unverändert übernommen. Viele der verwendeten Knoten wurden im Rahmen eines Fortgeschrittenen-Praktikums entwickelt [65]. Alle für die Hauptanwendung relevanten Knoten werden in diesem Kapitel beschrieben (siehe Abschnitt 1.3). Auf Knoten, die unverändert übernommen wurden, wird explizit hingewiesen.

5.2.1 Quellknoten

5.2.1.1 CDDANode

Der CDDANode ist nicht im Rahmen dieser Arbeit entstanden, sondern wurde schon für den Multimedia-Box Prototypen entwickelt und wird hier unverändert übernommen [65].

Der CDDANode liest eine Audio-CD digital aus, dabei greift er auf Funktionen der Paranoia-Library [14] zu. Die Daten auf einer Audio-CD sind immer mit 44100Hz und 16 Bit gesampelt für den rechten und linken Kanal. Das Ausgabeformat des Knotens ist deshalb auf dieses Format beschränkt.

Neben dem reinen Auslesen der Audiodaten, bietet der Knoten die Möglichkeit, die Anzahl der Tracks einer CD abzufragen, einzelne Tracks anzuspringen und das Vor- und Zurückspulen innerhalb eines Tracks. Zum Spulen wird dem Knoten das Event *seek* geschickt, das die Anzahl Sekunden enthält, die der Knoten überspringen soll, dabei wird auch die Trackgrenze berücksichtigt, d.h. wird über das Ende eines Tracks hinausgespult, so wird mit dem nächsten Track fortgesetzt.

Die CD-Tracks sind durchnummeriert, der Knoten kann aber bei Bedarf Interpret und Namen der einzelnen Lieder über das Internet abfragen. Dieser Service wird von FreeDB [24] angeboten.

5.2.1.2 DVBReadNode

Auch der DVBReadNode ist nicht im Rahmen dieser Arbeit entstanden. Er wurde schon für den Multimedia-Box Prototypen eingesetzt [65].

Der DVBReadNode empfängt mit der in Abschnitt 4.2.8 vorgestellten DVB-Karte, digitale TV-Programme über Satellit oder Kabel und schickt sie als MPEG Videostrom weiter. Dabei wird auch unterschieden, ob die eingesetzte DVB-Karte

einen eingebauten MPEG2 Dekoder hat, der optional eingeschaltet werden kann. Der Knoten kann deshalb zwei unterschiedliche Ausgangsformate haben:

- Einzelbilder im YV12 Format.
- MPEG2 PES Strom, der die komprimierten Audio- und Videodaten enthält, so wie sie über Satellit empfangen wurden [44].

Die YV12 Bilder werden über das Video4Linux [49] Interface empfangen. Das Umschalten auf verschiedene Kanäle und das Erzeugen eines MPEG2 PES Stroms wird mit Hilfe von Funktionen, die aus dem VDR [46] übernommen wurden, realisiert. Zur Konfiguration des Knotens werden die vom VDR benutzten Konfigurationsdateien (channels.conf und setup.conf) verwendet. Hier werden u.a. die Frequenzen und Namen der einzelnen Sender gespeichert. Die Beschreibung der Dateien befindet sich im Anhang A.4.

5.2.1.3 MP3ReadNode

Der MP3ReadNode liest eine MP3 Datei und verschickt sie als NMM-Buffer. Der Name der Datei kann dem Knoten über das Event *usefile* mitgeteilt werden. Nachdem der Knoten die Datei geöffnet hat, prüft er das Format, das u.a. die Anzahl der Kanäle (Mono/Stereo) und die Samplefrequenz enthält, und verschickt es als In-Stream Event.

Der MP3ReadNode wurde für den Multimedia-Box Prototypen [65] entwickelt und wird für diese Arbeit unverändert übernommen.

5.2.1.4 PNGReadNode

Der PNGReadNode liest eine Datei im “Portable Network Graphics” [76] Format und verschickt das enthaltende Bild als NMM-Buffer. Das Ausgangsformat ist immer RGB, unabhängig von dem Pixelformat der Eingangsdatei. Sie kann ein Palettenformat haben mit 2, 16 usw. Farben oder ein Graustufenformat. Die Konvertierung nach RGB übernimmt die verwendete PNG-Library [30]. Der Dateinamen wird per *usefile* Event an den Knoten übermittelt.

Der PNGReadNode wurde in [65] entwickelt und wird für diese Arbeit unverändert übernommen.

5.2.1.5 DVDNavReadNode

Auf einer DVD (Digital Versatile Disc) können beliebige Daten gespeichert werden. Eine (einseitigen) DVD kann dabei bis zu 8 Gigabyte Daten speichern. Auf Grund der hohen Kapazität wird sie im Unterhaltungsbereich aber hauptsächlich für die Speicherung von Videos verwendet. Die Datenrate einer solchen Video-DVD liegt bei maximal 11,08 Mbps. Durchschnittlich liegt die Datenrate bei ca. 4 Mbps, so dass über 4 Stunden Video und Audio gespeichert werden können. Neben

diesen MPEG2 komprimierten Videodaten können bis zu 8 Dolby Digital (AC3) Audiospuren und bis zu 32 Untertitel vorhanden sein.

Zusätzlich kann eine Video-DVD mit einem Menü ausgestattet werden, in dem man Bonusmaterial oder einzelne Szenen des Films auswählen kann oder Ton und Untertitel einstellen kann. Die Navigation geschieht in der Regel mit einer Fernbedienung, die Tasten für hoch, runter, links, rechts und eine Bestätigungstaste ähnlich der “Enter”-Taste der PC-Tastatur besitzt. Die Video-DVD ist aufgeteilt in ein oder mehrere Titel (engl. title), jeder Titel kann wiederum ein oder mehrere Kapitel (engl. chapter) enthalten. Dabei sind Titel ganze Filme oder Episoden und Kapitel sind Teile der Titel, wie eine oder mehrere Szenen.

“Seamless Playback” erlaubt es, dass ohne Unterbrechung des Videos an alternative Stellen gesprungen werden kann, wie etwa einem alternativen Ende oder eines Direcotors-Cut. Eine detaillierte Beschreibung jedes Features der DVD kann in [42] nachgelesen werden.

Der DVDNavReadNode kann mit Hilfe der libdvdnav-Library eine DVD Laufwerk ansteuern, um damit eine Video-DVD zu lesen [48]. Bei den gelesenen Daten handelt es sich einerseits um MPEG-Program-Stream (MPEG PS-Stream) Daten und andererseits um Events, die u.a. die Navigationsdaten enthalten [42]. Der MPEG PS-Stream enthält die eigentlichen Audio- und Videodaten, die gemultiplext gespeichert sind, d.h. Audio- und Videodaten werden nicht getrennt übertragen, sondern sie wechseln sich im Datenstrom ab. Sie werden in Paketen gespeichert, die eine eindeutige Nummer für Audio, Video und zusätzliche Daten haben, damit der Demultiplexer, also der Teil, der den gemultiplexten Strom wieder in Audio- und Videoteile zerlegt, feststellen kann, welches Paket zu welchem Strom gehört. Neben diesen Audio- und Videoteilen befinden sich Untertitel (SPU, Sub Picture Unit), Präsentations-Kontroll Informationen (PCI, engl. presentation controll information) und Daten-Suchinformationen (DSI, engl. data search information) in einem DVD Datenstrom.

Audio, Video und Untertitel können mit dem MPEGDemuxNode gedemultiplext werden (siehe Abschnitt 5.2.2.4). Der DVDNavReadNode verwendet die libdvdnav [48], um Daten von der DVD zu lesen, PCI und DSI Pakete zu demultiplexen, auszuwerten und um auf eingehende Events wie Tastendrücke oder Kapitelwechsel zu reagieren. Die ausgewerteten PCI und DSI Pakete werden als NMM-Events weitergeleitet.

Menüs und Knöpfe

Informationen über den Aufbau des DVD-Menüs und das Aussehen und die Anordnung der Knöpfe stehen in den PCI-Paketen. Es können bis zu 36 rechteckige Knöpfe auf dem Bildschirm positioniert werden. Knöpfe können Transparenzen enthalten, so dass beliebige Formen möglich sind.

Menüknöpfe sind eng mit Untertiteln verbunden. Es gibt Vorder- und Hintergrundpixel, die benutzt werden, um Knöpfe auf einen Videostrom zu zeichnen. Knöpfe können aber auch auf unbewegte Einzelbilder gezeichnet werden. Solche

Einzelbilder werden als “Still Frame” bezeichnet. Unsichtbare oder transparente Knöpfe können gezeichnet werden, indem der Pixelkontrast gesetzt wird. Nähere Informationen dazu befinden sich in Abschnitt 5.2.2.2.

Consumer DVD-Player werden mit einer Fernbedienung gesteuert, d.h. um einzelne Menüpunkte zu selektieren (highlighten). Menüpunkte werden durch Knöpfe realisiert. Jeder Knopf speichert dabei Informationen, welcher Knopf selektiert werden soll, wenn man die entsprechende Taste auf der Fernbedienung drückt. Wird ein Knopf selektiert oder aktiviert, wird er mit einer anderen Farbe und anderem Kontrast gezeichnet (siehe Abschnitt 5.2.2.2). Beim Aktivieren einen Knopfes wird ein oder mehrere Kommandos ausgeführt, sie werden von einer Virtual Machine interpretiert, die Teil der libdvdnav ist. Bemerkbar machen sich vorallem solche Kommandos, die einen Menüwechsel oder Szenenwechsel bewirken.

Ein- und ausgehende Events

Der DVDNavReadNode reagiert auf folgende NMM-Events:

- **device** *(string device)* Setzt den Pfad zum DVD-Laufwerk, als Standard wird “/dev/cdrom” benutzt.
- **selectDVDChapter** *(int title, int chapter, int angel)* Springt zum angegebenen Kapitel im angegebene Titel. Die Auswertung der Winkel-Information ist in der aktuellen Version noch nicht implementiert und wird daher ignoriert.
- **dvd_button_up** Führt die Aktion aus, die mit der Navigationstaste “nach oben” verknüpft ist, im DVD-Menü wird meist ein Knopf nach oben gesprungen und dieser dann selektiert.
- **dvd_button_down** Analog wie *dvd_button_up*
- **dvd_button_left** Analog wie *dvd_button_up*
- **dvd_button_right** Analog wie *dvd_button_up*
- **dvd_button_press** Aktiviert einen selektierten Knopf, es wird danach meist ein Menü- oder Szenenwechsel durchgeführt.

Die libdvdnav analysiert den gelesenen DVD-Datenstrom und liefert daraus zusätzliche Informationen. Diese Informationen werden im DVDNavReadNode als NMM-Events verpackt:

- **disable_overlay** Wird geschickt, um dem OverlayNode (Abschnitt 5.2.2.3) mitzuteilen, dass das momentan übergeblendete Bild ausgeblendet werden soll. Dieses Event tritt dann auf, wenn man vom Menü in den Hauptfilm wechselt. Menüknöpfe werden dadurch ausgeblendet.

- **still_frame** kennzeichnet ein Menü, in dem kein bewegter Hintergrund vorhanden ist. Dieses Event ist wichtig, da keine Daten zum Videodecoder gelangen solange ein starres Bild angezeigt wird. Hierdurch hat der OverlayNode die Möglichkeit das letzte empfangene Bild zwischenzuspeichern, damit er bei einer Knopfselektion das Overlay richtig durchführen kann (siehe Abschnitt 5.2.2.3).
- **spu_palette_entry** *(unsigned int color_nr, unsigned int color)* Kennzeichnet eine Änderung eines Eintrags in der Farbpalette für Knöpfe und Untertitel. “color_nr” ist die Nummer der Farbe in der Palette, “color” ein neuer YUV Farbwert (siehe Abschnitt 5.2.2.2).
- **spu_color_scheme** *(unsigned int scheme_nr, unsigned int selection_color, unsigned int activation_color)* Überträgt ein Farbschema mit der angegebenen Nummer (scheme_nr), dabei ist die Selektionsfarbe (selection_color) und die Aktivierungsfarbe (activation_color) ein Eintrag aus der Farbpalette (siehe Abschnitt 5.2.2.2).
- **dvd_button_number** *(unsigned int Buttons)* Überträgt die Anzahl Knöpfe eines Menüs.
- **dvd_button_coord** *(unsigned int xs, unsigned int ys, unsigned int xe, unsigned int ye)* Überträgt die Maße eines Rechtecks, das einen Knopf beschreibt. Dabei ist (xs,ys) die obere linke Ecke und (xe,ye) die untere rechte Ecke. In Abschnitt 5.2.2.2 wird beschrieben, woher die Pixeldaten dieses Rechtecks genommen werden.
- **dvd_button_scheme** *(unsigned int scheme)* Enthält die Nummer des Farbschemas, das für das Zeichnen der Knöpfe benutzt werden soll.
- **set_spu_stream** *(unsigned int spu_nr)* Überträgt die Nummer des Untertitelstroms, der angezeigt werden soll. Sie wird immer dann gesendet, wenn die Sprache der Untertitel wechselt.
- **set_audio_stream** *(unsigned int audio_nr)* Überträgt die Nummer des Audiostroms, der dekodiert werden soll. Sie wird dann gesendet, wenn die Sprache sich ändert.

5.2.2 Verarbeitungsknoten

5.2.2.1 OSDManagerNode

In Abschnitt 5.1 wurde ein grafisches Benutzerinterface vorgestellt. Die Darstellung dieses Benutzerinterfaces wird durch den OSDManagerNode (On-Screen-Display Manager) realisiert. Die Schnittstelle zwischen dem grafisches Benutzerinterface und dem OSDManagerNode sind dabei *Windows*, die unterschiedliche Widgets enthalten können (vgl. Abschnitt 5.1.1). Windows erlauben den direkten Zugriff auf die Pixeldaten einzelner GUI-Komponenten.

Neben der Darstellung aller vorhandenen Windows bietet der OSDManagerNode eine Schnittstelle, mit der man leicht zusätzliche Informationen wie Zeit, Fortschrittsbalken und Statusinformationen anzeigen kann. Auch die Darstellung und Navigation einer einfachen Messagebox ist im Knoten implementiert. Der OSDManagerNode soll alle Informationen über einen laufenden Videostrom blenden können. Dieses "Overlaying" kann sehr rechenintensiv sein, vorallem wenn man Grafiken mit Transparenzeffekten benutzt. Damit das Overlaying auch auf schwächeren Prozessoren ausgeführt werden kann, sind einige Optimierungen nötig.

Mit dem OSDManagerNode können auch Zeichensätze verwaltet werden. Zeichensätze können mit einer eindeutigen ID zum OSDManagerNode hinzugefügt werden und jedes Objekt, das Zugang zum OSDManagerNode hat, kann auf diese Zeichensätze zugreifen.

Der OSDManagerNode ist ein Multiplexer-Knoten und hat zwei Eingänge. Ein Eingang wird für den Videostrom benutzt, auf dem das Overlay durchgeführt werden soll, der andere Eingang ist optional. Dieser zweite Eingang wird benutzt, um Zeitstempel eines Buffers auszuwerten und anzuzeigen. Bei einem laufenden Videostrom am 1. Eingang wird dieser 2. Eingang nicht benötigt, da die Buffer automatisch die richtigen Zeitstempel enthalten (Abbildung 5.24). Soll die genaue Zeitinformation eines Audiostroms anzeigen werden, so wird an Eingang 1 ein Standbild angelegt (Abschnitt 5.2.1.4) und an Eingang 2 der Audiostrom, dessen Zeitstempel extrahiert und angezeigt werden sollen (Abbildung 5.25).

Zum Hinzufügen und löschen eines Window, gibt es entsprechende Methoden im OSDManagerNode:

```
void addWindow( Window* win );
void delWindow( Window* win );
```

Dabei ist `win` ein Zeiger auf ein Window (siehe Abschnitt 5.1.1). Der OSD-Manager blendet jedes hinzugefügte Window über das eingehende Videobild, dabei werden auch Transparenzen berücksichtigt (Abbildung 5.26). Da Videodaten oft im YV12 Farbformat gespeichert sind, müssen auch die Pixeldaten der Windows in diesem Format vorliegen. Deshalb werden die in Abschnitt 5.1.1 erwähnten YV12 Windows verwendet, die auch Transparenzinformationen speichern können. Die

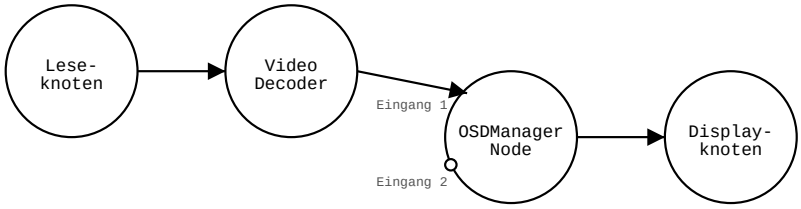

Abbildung 5.24: Hier wird nur ein Eingang des OSDManagerNode benutzt, da Zeitinformationen direkt aus den Video-Buffern extrahiert werden können.

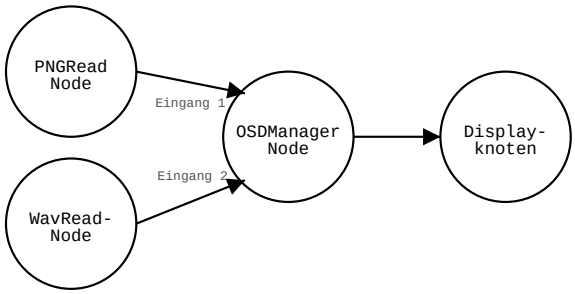

Abbildung 5.25: OSDManagerNode mit zwei verbundenen Eingängen. An Eingang 1 hängt ein PNGReadNode, der nur einmal ein Bild liefert, an Eingang 2 hängt ein WavReadNode, der Audiodaten liest und ständig an den OSDManagerNode schickt, der die Zeitstempel extrahiert.

Konvertierung der Pixeldaten der Windows in das YV12 Farbformat wird aber nur dann durchgeführt, wenn sich der Inhalt des Windows geändert hat.

Overlay

Der Begriff “Overlay” bezeichnet in der Bildverarbeitung das Überblenden eines Bildes auf ein anderes. Das übergeblendete Bild kann dabei das darunterliegende komplett verdecken oder es kann zu einem bestimmten Grad durchscheinen. Wieviel ein Bild durch ein anderes durchscheinen kann wird durch die *Transparenz* bestimmt. Je höher die Transparenz eines Bildes desto mehr kann ein darunterliegendes Bild durchscheinen.

Transparente Grafiken werden oft in Einstellungsmenüs von Fernsehgerät und Satelitten-Receiver (Settop-Box) benutzt. In diesen Menüs scheint das laufende Videobild durch. Auch die Multimedia-Box soll die Möglichkeit besitzen, transparente Grafiken über das laufende Videobild zu blenden, um so gleichzeitig in

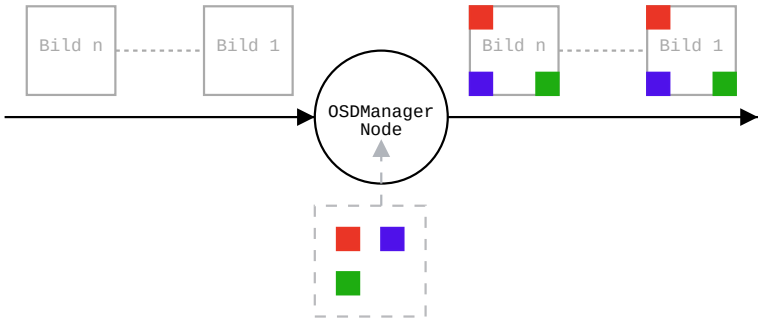

Abbildung 5.26: Der OSDManagerNode mischt alle eingehenden Videobilder mit allen sichtbaren Windows.

Menüs zu navigieren während ein Video abgespielt wird.

Das Überblenden von einem Bild auf ein anderes ist u.a. Aufgabe des OSD-ManagerNode. Die Formel, um das Overlay von Bildern im RGB Format durchzuführen lautet [10]:

$$out = alpha * in_0 + (1 - alpha) * in_1$$

Dabei ist in_0 und in_1 eine Farbkomponente (R,G oder B) im Bereich 0 bis 1 und $alpha$ ist die Intensität, mit der Farbkomponente in_0 auf in_1 gezeichnet wird, d.h. bei einem Alphawert von 1 wird in_1 komplett durch in_0 ersetzt. Da die Konvertierung von RGB nach YV12 nur aus einer Matrizenmultiplikation besteht (siehe [65]), kann diese Formel auch für Bilder im YV12 Format benutzt werden. Pixel und Alphawerte haben einen Wertebereich von 0 bis 255 haben, deshalb muss die Formel noch entsprechend angepasst werden:

$$out = (alpha * in_0 + (255 - alpha) * in_1)/255$$

Jeder Pixel hat einen Alphawert, wenn zwei Pixel überblendet werden, muss auch der resultierende Pixel einen Alphawert besitzen. Er wird berechnet, indem man obige Formel benutzt und dem $alpha$ Wert die volle Intensität gibt:

$$alpha = (255 * alpha_1 + (255 - alpha_1) * alpha_0)/255$$

Mit obiger Formel kann eine C-Funktion implementiert werden, die alle Pixel zweier Bilder kombiniert. Schwächere Prozessoren sind schnell überfordert, wenn sie für 25 Bilder (PAL) oder 29,97 Bilder (NTSC) pro Sekunde, die man für eine flüssige Darstellung eines Videostroms braucht, das Overlay durchführen sollen. Deshalb wird eine speziell optimierte Assembler-Version eingesetzt, die mit Hilfe von MMX Befehlen moderner Prozessoren das Overlay erledigt. Ein Überblick über die Assembler-Befehle der Intel-CPUs befindet sich in [12]. MMX Optimierung wird im wesentlichen benutzt, um die R,G,B,A Komponenten zu berechnen. Ziel ist die Umsetzung folgenden C-Codes in MMX, der ein Overlay zwischen zwei RGBA-Pixel durchführt:

```
unsigned int r1, r2, r3;
unsigned int g1, g2, g3;
unsigned int b1, b2, b3;
unsigned int a1, a2, a3;

r1 = pixel_0[0];
g1 = pixel_0[1];
b1 = pixel_0[2];
a1 = pixel_0[3];

r2 = pixel_1[0];
g2 = pixel_1[1];
b2 = pixel_1[2];
```

```
a2 = pixel_1[3];

r3 = (a1*r1 + (255-a1) * r2)/255;
g3 = (a1*g1 + (255-a1) * g2)/255;
b3 = (a1*b1 + (255-a1) * b2)/255;
a3 = (a2*255 + (255-a2) * a1)/255;
```

MMX Befehlssatz

Im folgenden werden die MMX Befehle aufgelistet, die für das Overlay benötigt werden. Eine vollständige Auflistung aller MMX-Befehle befindet sich in [38, 37].

Ein MMX fähiger Prozessor hat zusätzliche 8 MMX-Register, die jeweils 64 Bit breit sind. Bezeichnet werden sie mit mm0 - mm7. MMX Befehle haben ein oder zwei Operanden, die wie nachfolgend bezeichnet werden:

- **imm8** Angabe eines direkten Bytewertes, imm8 ist eine vorzeichenbehaftete Zahl zwischen -128 und +127.
- **r/m32** 32 Bit Register oder Speicheroperand
- **mm/m32** bezeichnet die unteren 32 Bits eines MMX Registers oder einer Speicherzelle
- **mm/m64** bezeichnet ein 64 Bit MMX Register oder eine 64 Bit Speicherzelle

Zur Umsetzung des C-Codes in MMX werden im wesentlichen folgende MMX Befehle benutzt:

- **MOVD** *mm, r/m32* Kopiert 32 Bits eines Register/Speicher in ein MMX Register
- **MOVQ** *mm, mm/m64* Kopiert 64 Bit eines MMX Registers/Speicher in ein MMX Register
- **PAND** *mm, mm/m64* Bitweise UND-Verknüpfung von 64 Bit eines MMX Registers/Speicher mit einem MMX Register

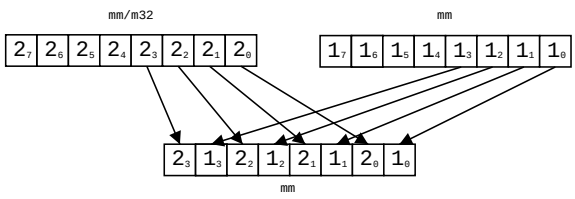

Abbildung 5.27: Abwechselnd werden die unteren Bytes der MMX Quellregister in ein Zielregister geschrieben

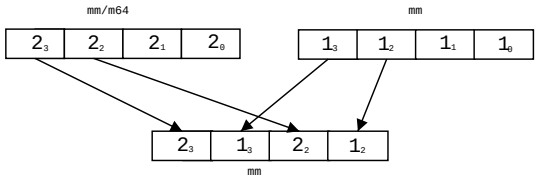

Abbildung 5.28: Abwechselnd werden die oberen Worte der MMX Quellregister in ein Zielregister geschrieben

- **POR** *mm, mm/m64* Bitweise ODER-Verknüpfung von 64 Bit eines MMX Registers/Speicher mit einem MMX Register
- **PADDW** *mm, mm/m64* Addiert ein packed Word eines MMX Registers/-Speicher zu einem packed Word eines MMX Registers. Bei einem packed Word werden die 64 Bits eines MMX Registers nicht als ganzes betrachtet, es wird aufgeteilt in vier 16 Bit Teile, die unabhängig addiert werden, d.h. hier werden acht 16 Bit Werte parallel addiert.
- **PSUBW** *mm, mm/m64* Subtrahiert ein packed Word eines MMX Registers/-Speicher von einem MMX Register
- **PMULW** *mm, mm/m64* Multipliziert ein packed Word eines MMX Registers mit dem packed Word eines MMX Registers/Speicher, die unteren 16 Bit des Ergebnisses werden im MMX Register gespeichert.
- **PSLLW** *mm, imm8* Shiftet die vier 16 Bite Werte eines MMX Registers um imm8 nach links, dabei werden Nullen aufgefüllt.
- **PSRLW** Shiftet die vier 16 Bite Werte eines MMX Registers um imm8 nach rechts, dabei werden Nullen aufgefüllt.
- **PUNPCKLBW** *mm, mm/m32* Kopiert abwechselnd die unteren 4 Bytes eines MMX Registers und eines zweiten MMX Registers oder Speicher in ein MMX Register (Abbildung 5.27).

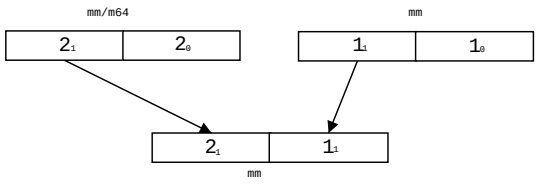

Abbildung 5.29: Oberes Doublewort zweier Register werden in ein Zielregister geschrieben

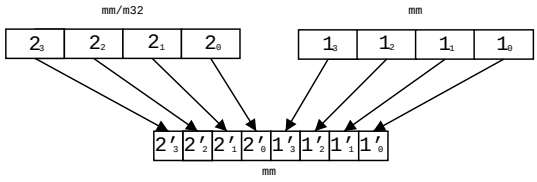

Abbildung 5.30: Alle 16 Bit vorzeichenbehaftete Worte der zwei Quellregister werden in einen vorzeichenlosen 8 Bit Wert konvertiert und in das Zielregister geschrieben

- **PUNPCKHWD** *mm, mm/m32* Kopiert abwechselnd die oberen 2 Worte eines MMX Registers und eines zweiten MMX Registers oder Speicher in ein MMX Register (Abbildung 5.28).
- **PUNPCKHDQ** *mm, mm/m32* Kopiert das oberen Doublewort eines MMX Registers und eines zweiten MMX Registers/Speicher in ein MMX Register (Abbildung 5.29).
- **PACKUSWB** *mm, mm/m64* Kopiert vier vorzeichenbehaftete Worte eines MMX Registers und eines zweiten MMX Registers/Speicher in ein MMX Register, dabei werden die Worte auf einen vorzeichenlosen Bytewert reduziert, tritt ein Überlauf auf, wird ein Wert von 0xff benutzt, bei einem Unterlauf 0x00 (Abbildung 5.30).

Overlay mit MMX Optimierung

Mit Hilfe der gezeigten MMX Befehle, lässt sich der C-Code Ausschnitt, der das eigentliche Overlay durchführt, in Assembler umsetzen. Dabei wird die oben gezeigt Overlayformel ein wenig umgeformt:

$$out = (alpha * (in_0 - in_1) + 256 * in_1)/256$$

Der Wertebereich wurde auf 256 erhöht, da Division und Multiplikation mit Zweierpotenzen durch Bitschiebeoperationen optimiert werden können. Folgender Assemblercode erzielt das gewünschte Ergebnis:

```
1)  pxor      mm5, mm5  ; mm5 = 0

    ; mm6 = 0100000000000000
2)  movq      mm6, 0x0100000000000000
    ; mm7 = 0000ffffffffffff
3)  movq      mm7, 0x0000ffffffffffff

4)  movd      mm0, [esi] ; mm0 = 00000000A1B1G1R1
5)  movd      mm1, [edi] ; mm1 = 00000000A2B2G2R2
```

```
 6)  punpcklbw mm0, mm5   ; mm0 = 00A100B100G100R1
 7)  punpcklbw mm1, mm5   ; mm1 = 00A200B200G200R2

 8)  movq      mm2, mm0
 9)  punpckhwd mm2, mm2   ; mm2 = 00A100A1xxxxxxxx
10)  punpckhdq mm2, mm2   ; mm2 = 00A100A100A100A1

11)  pand      mm0, mm7
12)  por       mm0, mm6   ; mm0 = 100000B100G100R1

13)  psubw     mm0, mm1

14)  pmullw    mm0, mm2

15)  psllw     mm1, 8
16)  paddw     mm0, mm1

17)  psrlw     mm0, 8

18)  packuswb  mm0, mm0   ; mm0 = 00000000A3B3G3R3

19)  movd     [edi], mm0
```

Zeile 1-3 initialisieren die MMX Register mm5 mit 0 (durch XOR-Verknüpfung mit sich selbst), mm6 und mm7. Zur einfachen Darstellung wurde in mm6 und mm7 einfach konstante 64 Bit Werte hineingeschrieben, was normalerweise nicht möglich ist. Stattdessen muss man einen 64 Bit Wert aus einer Speicherzelle lesen.

Es wird angenommen, dass die normalen Prozessorregister “esi” und “edi” Pointer auf RGBA Pixeldaten enthalten. In Zeile 4 und 5 wird jeweils ein 32 Bit breiter Pixel in Register mm0 und mm1 geladen:

```
mm0 = 00 00 00 00 A1 B1 G1 R1
mm1 = 00 00 00 00 A2 B2 G2 R2
```

Zu Beachten ist hier, dass die Bytes in umgekehrter Reihenfolge als im Speicher abgelegt werden.

In Zeile 6 und 7 werden die vorher 8 Bit breite Farbkomponenten R,G,B und A auf 16 Bit erweitert, da bei nachfolgenden Multiplikationen so ein Überlauf abgefangen wird. Die Erweiterung auf 16 Bit wird mit dem PUNPCKLBW Befehl (Abbildung 5.27) durchgeführt, indem man als zweites Register, ein Register mit einem Nullwert übergibt:

```
mm0 = 00 A1 00 B1 00 G1 00 R1
mm1 = 00 A2 00 B2 00 G2 00 R2
```

Laut Formal werden alle Farbekomponenten mit einem Alphawert multipliziert, deshalb muss der Alphawert A1 genau wie die Farbkomponenten vier mal in ein MMX Register geschrieben werden. Das wird erreicht, indem man die Befehle PUNPCKHWD (Abbildung 5.28) und PUNPCKHDQ (Abbildung 5.29) hintereinander schaltet, beide mal mit den gleichen Registern (Zeile 8-10):

```
mm2 = 00 A1 00 A1 00 A1 00 A1
```

Der Alphawert A1 muss jetzt in MMX Register mm0 durch den Wert 256 ersetzt werden, das wird durch die am Anfang vorgestellten Register mm6 und mm7 realisiert, indem man den Alphawert mit einer UND-Verknüpfung erst löscht und mit einer ODER-Verknüpfung den neuen Wert hineinschreibt (Zeile 11-12):

```
mm0 = 10 00 00 B1 00 G1 00 R1
```

Die Daten liegen jetzt so in den Registern, dass man alle Farbkomponenten, mit jeweils einem MMX Befehl bearbeiten kann. Zuerst müssen die Pixelwerte subtrahiert werden (Zeile 13):

```
mm0 = (256-A2) | (B1-B2) | (G1-G2) | (R1-R2)
```

Dann kann mit dem Alphawert A1 multipliziert werden (Zeile 14):

```
mm0 = A1*(256-A2) | A1*(B1-B2) | ...
      A1*(G1-G2) | A1*(R1-R2)
```

Jetzt muss noch der 2. Pixelwert mit 256 multipliziert werden (Linksshift von 8) und addiert werden (Zeile 15-16):

```
mm0 = A1*(FF-A2)+256*A2 | A1*(B1-B2)+256*B2 | ...
      A1*(G1-G2)+256*G2 | A1*(R1-R2)+256*R2
```

Nun müssen die Werte wieder durch 256 dividiert (Rechtsshift von 8, Zeile 17):

```
mm0 = 00 A3 00 B3 00 G3 00 R3
```

Die 16 Bit Werte dieses Registers müssen jetzt wieder in 8 Bit Werte konvertiert werden, das geschieht mit dem PACKUSWB Befehl, indem man Ziel und Quellregister gleich wählt:

```
mm0 = 00 00 00 00 A3 B3 G3 R3
```

Das Ergebnis kann jetzt in eine 32 Bit breite Speicherzelle geschrieben werden.

Schnittstellen des OSDManagerNode

Neben dem Hinzufügen von Windows und dem Überblenden von Bildern, kann der OSDManagerNode bestimmte Aktionen auszuführen, dabei greift er auf die in Abschnitt 5.1.8 gezeigten Widgetunits zurück, d.h. er leitet zum Beispiel Anfragen zum Darstellen einer Messagebox direkt an die entsprechende Widgetunit weiter. Über das Interface können die Widgetunits gesteuert werden:

Result displayMessage(string headerstr, string text, string label1, string label2, string label3, unsigned int sel) Erzeugt eine Messagebox mit Hilfe der MessageUnit.

Result messageLeft() Navigiert in der Messagebox nach links.

Result messageRight() Navigiert in der Messagebox nach rechts.

string messageID() Liefert die ID des gedrückten Knopfes der Messagebox.

Result deleteMessage() Löscht die Messagebox.

Result displayTime(string header, unsigned int x, unsigned int y) Stellt die Zeit mit Hilfe einer TimeUnit dar.

Result hideTime() Blendet die Zeitdarstellung aus.

Result displayProgressBar(string header, unsigned int x, unsigned int y) Zeigt einen Fortschrittsbalken mit Hilfe einer ProgressBarUnit.

Result hideProgressBar() Blendet den Fortschrittsbalken aus.

void setMaxProgress(unsigned int max) Setzt den maximalen Wert des Fortschrittbalkens.

void setProgress(unsigned int progress) Setzt den aktuellen Wert des Fortschrittbalkens.

void updateProgress() Erzwingt das Neuzeichnen des Fortschrittbalkens.

void setFont(string id, BitmapFont* font) Fügt dem OSD-Manager einen Zeichensatz mit einer ID hinzu, damit u.a. die Messagebox angezeigt werden kann, muss mindestens ein Zeichensatz mit der ID “medium” hinzugefügt werden.

void setFontList(map<string, BitmapFont*> f) Übergibt dem OSD-Manager eine Liste mit Zeichensätzen und deren ID.

BitmapFont* getFont(string id) Liefert den Zeichensatz mit der angegebene ID zurück.

void setBitmapReader(BitmapReader* br) Übergibt einen BitmapReader, aus dem u.a. die Icons für die Messagebox genommen werden.

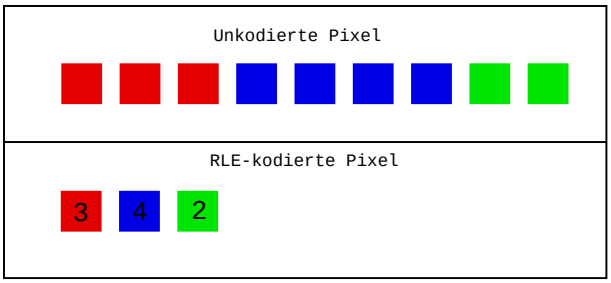

Abbildung 5.31: Bei einem RLE-kodierten Bild werden nicht alle Pixel einzeln gespeichert, sondern bei gleichen Pixeln, die nacheinander folgen, wird einmal der Farbwert gespeichert und deren Anzahl.

5.2.2.2 SPUDecodeNode

Der SPUDecodeNode dekodiert komprimierte Pakete eine SPU-Stroms. Dieser “Sub Picture Unit” Strom enthält Daten zur Anzeige der Untertitel und Menüknöpfe einer Video-DVD. Eine Video-DVD kann bis zu 32 Untertitel (SPU) Ströme enthalten, die über das laufende Videobild geblendet werden. Neben den eigentlichen Untertiteln, werden SPU Ströme dazu verwendet, Menüs oder einfache Animationen aufzubauen. Es handelt sich dabei um einfache Bitmapgrafiken, die fast die volle Größe eines Videobildes haben können. Die Bitmapgrafiken sind run-length encoded (RLE), d.h. es wird nicht jeder Pixel explizit gespeichert, sondern mehrere gleiche Pixel werden zusammengefasst und ihre Anzahl gespeichert (vgl. Abbildung 5.31).

Jedes Pixel ist 2 Bits groß, somit kann eine Bitmap maximal aus 4 Farben bestehen. Die einzelnen Farben werden mit Background, Pattern/Foreground, Emphasis-1 und Emphasis-2 bezeichnet. Der 2 Bit breite Pixelwert gibt die Farbe nicht direkt an, da man sonst wirklich nur auf 4 konstante Farben beschränkt wäre. Die konkrete Farbzuweisung läuft in zwei Schritten. Zuerst wird die Nummer des Pixelwertes (0-3) der Bitmap in einer Zwischentabelle nachgeschlagen, dort bekommt der Wert einen Alphawert und einen Paletteneintrag (0-15) zugewiesen. In einer Farbpalette wird dann der konkrete Farbwert nachgeschlagen (Abbildung 5.32). Ein Alphawert von 0 bedeutet, das der Pixel komplett transparent ist, ein Wert von 15 zeigt den Pixel ohne Transparenz. Die Farbpalette enthält 16 Einträge mit jeweils 24 Bit YUV Farbwerten.

Die Größe der Bitmap kann fast die Größe eines Videobildes haben, d.h. eine Maximalgröße von 720x478 Pixeln bei NTSC und 720x573 bei einem PAL Bild. Die Bitmapgröße, deren Inhalt und die Farb- und Alphawerte können in jedem einzelnen Videoframe verändert werden.

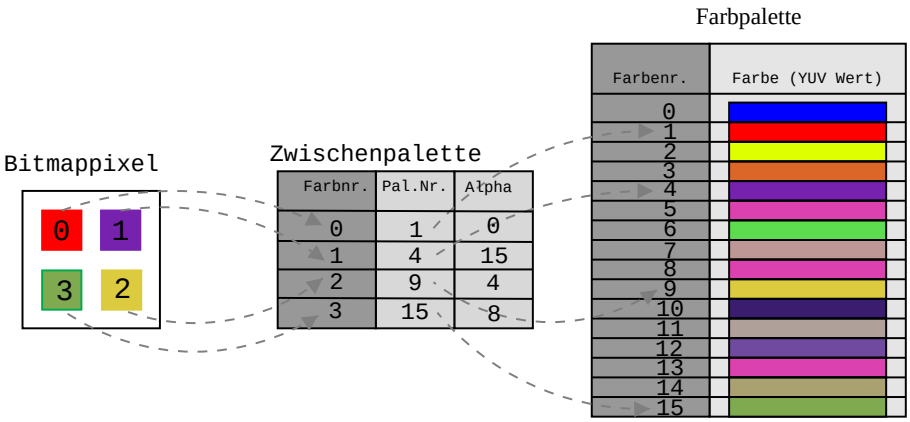

Abbildung 5.32: Pixelfarben werden indirekt angegeben. Der Pixelwert (0-3) einer Bitmap bekommt in einer Zwischenpalette einen Alphawert und einen Palettenteintrag von 0-15 zugewiesen, der wiederum bekommt in einer Farbpalette den konkreten Farbwert zugewiesen.

Besonderheiten bei Menüknöpfen

Bei näherer Betrachtung einer Video-DVD fällt auf, dass die Menüs stark variieren können. Es gibt “einfache” Menüs, deren Knöpfe statisch sind und andere, bei denen anscheinend Videosequenzen in den Knöpfen abgespielt werden. Tatsächlich gibt es hier aber keinen Unterschied, denn auch die Menüs mit all ihren Knöpfen sind in einem Videostrom verpackt. Dieser Videostrom enthält die Knöpfe in ihrem unselektierten Zustand. Das bedeutet, auch ohne SPU-Decoder werden Menüs angezeigt. Jedoch bekommt man dann keinen Feedback, welcher Knopf gerade selektiert oder gedrückt wurde.

In Menüs werden SPU-Ströme benutzt, um bestimmte Bereiche des Bildes hervorzuheben. Eine dekodierte Bitmap des SPU-Stroms enthält dabei das Aussehen des Hervorhebungsbereichs und alle Bereiche, die hervorgehoben werden können. Diese Bereiche repräsentieren die Selektion der Menüknöpfe. Das in Abschnitt 5.2.1.5 erwähnte Event *dvd_button_coord* des DVDNavReadNode enthält die Koordinaten und Maße eines solchen Hervorhebungsbereiches. Die Extraktion dieser Bereiche wird in Abschnitt 5.2.2.3 genauer erklärt.

Hauptunterschied der Menüknöpfe zu den Untertiteln ist, dass Menüknöpfe mehrere Farbschema haben können. Die in Abbildung 5.32 gezeigte Zwischenpalette ist bei Menüknöpfen erweitert bzw. es gibt mehrere davon (Abbildung 5.33). Hervorhebungsbereiche können zwischen 3 Farbschemata wählen. Diese Schemata haben Einträge für den “Select State”. Die Farb- und Alpha-Wertzuweisung dieser Einträge wird dann benutzt, wenn ein Knopf selektiert wird. Farb- und Alpha-Wertzuweisungen des “Action State” werden benutzt, wenn der selektierte Knopf gedrückt wird. Jeder Knopf kann diese Zuweisungen aus einer der 3 Farbschemata

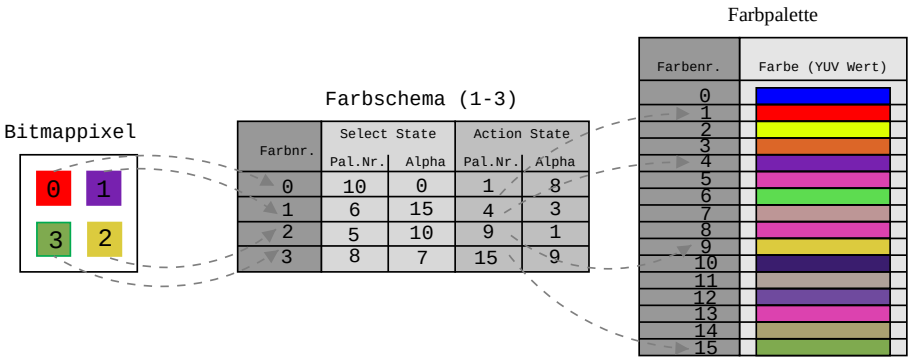

Abbildung 5.33: Bei der Hervorhebung von Menüknöpfen, kann zwischen 3 Farbschemata gewählt werden, jedes Schema hat Einträge für den Selektionszustand und Aktionszustand eines Knopfes

auswählen.

Aufbau der SPU Pakete

SPU Pakete werden im MPEG Strom mitübertragen, jedes SPU Paket hat eine Identifikationsnummer, die die Nummer des Untertitelstromes angibt. Es können insgesamt 32 verschiedene SPU-Ströme auf einer DVD vorhanden sein.

Nachdem die SPU-Pakete aus dem DVD-Datenstrom extrahiert wurden (siehe Abschnitt 5.2.2.4), werden sie im SPUDecodeNode gesammelt, bis genügend Daten vorhanden sind, um eine Bitmap zu dekodieren. Abbildung 5.34 zeigt eine solche vollständige Datenstruktur. Die ersten zwei Bytes geben die Länge der gesamten Datenstruktur an, sie kann demnach maximal 64 KBytes groß sein. Die nächsten beiden Bytes geben einen Offset zu einer Kontrollsequenz an, danach beginnt das eigentliche Datenpaket, das die Länge der Bitmapdaten enthält und die Daten im RLE Format selbst. Die Bitmapdaten sind getrennt in ungerade und gerade Zeilen. Im Datenpaket ist keinerlei Information über die Breite und Höhe der Bitmap, sie sind ein Teil der Kontrollsequenz. Die Kontrollsequenz enthält mehrere Kommandos, vor jedem Kommando steht ein Offset zum nächsten Kommando und die Startzeit, die angibt, wann das Kommando ausgeführt werden soll. Das Ende der Kontrollsequenz wird durch ein “0xFF” Byte signalisiert. Jedes Kommando wird durch einen 1 Bytewert repräsentiert, das abhängig vom Kommando weitere Parameter benötigt, wobei aber jedes Kommando eine feste Länge besitzt:

- **Kommando 0x00** *(1 Byte)* Kennzeichnet eine SPU, die für Menüs benutzt wird, also keinen Untertitel darstellt
- **Kommando 0x01** *(1 Byte)* Startzeit der Bitmapdarstellung
- **Kommando 0x02** *(1 Byte)* Stopzeit der Bitmapdarstellung

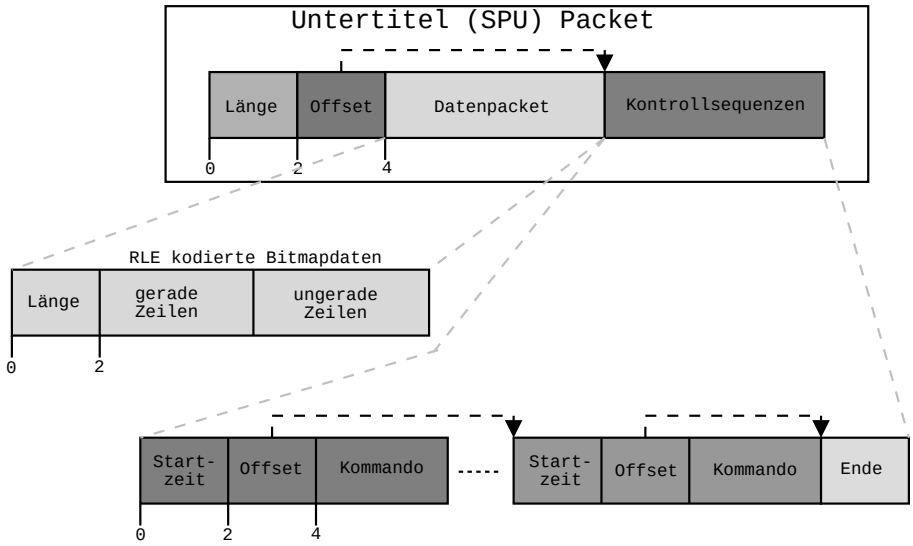

Abbildung 5.34: Aufbau eines SPU Pakets. Das Paket enthält einen Datenteil, in dem die RLE-kodierten Bitmapdaten abgelegt sind und einen Kontrollbereich, in dem mehrere Kommandos untergebracht sind. Kommandos können Anzeigedauer, Position und Farbe beeinflussen.

- **Kommando 0x03** *(4 Bytes)* Enthält die Farbzuordung der Zwischenpalette, die ersten 4 Bit enthalten die Information für Eintrag 0, usw.
- **Kommando 0x04** *(4 Bytes)* Enthält die Alphawertzuordung der Zwischenpalette
- **Kommando 0x05** *(7 Bytes)* Gibt Größe und Position der Bitmap an. Die ersten 12 Bit geben die X-Koordinate der oberen linken Ecke der Bitmap an, die nächsten 12 Bit geben die X-Koordinate der rechten unteren Ecke an, danach folgen 24 Bit, die die selbe Information für die Y-Koordinate enthalten.
- **Kommando 0x06** *(5 Bytes)* Die ersten 2 Bytes enthalten den Offset zu den geraden Zeilen der Bitmap im Datenpaket, die nächsten beiden enthalten den Offset zu den ungeraden Zeilen.

Der SPUDecodeNode reagiert auf einige Events, die der DVDNavReadNode (Abschnitt 5.2.1.5) verschickt. Hierzu zählen Events, die die Einträge der Farbpalette enthalten, die Farbschema und Zwischenpaletten enthalten und Angaben zu dem Status der Menüknöpfe besitzen. Der Knoten selbst trennt zwischen den reinen Untertitelströmen und den SPU Paketen, die Menüknopfbitmaps enthalten. Untertitel können direkt an den Nachfolgerknoten weitergeschickt werden ohne Zwischenspeicherung, da sie nur einmal dargestellt werden müssen. Menüknopfbitmaps hingegen werden solange im Knoten gespeichert, bis sie durch andere

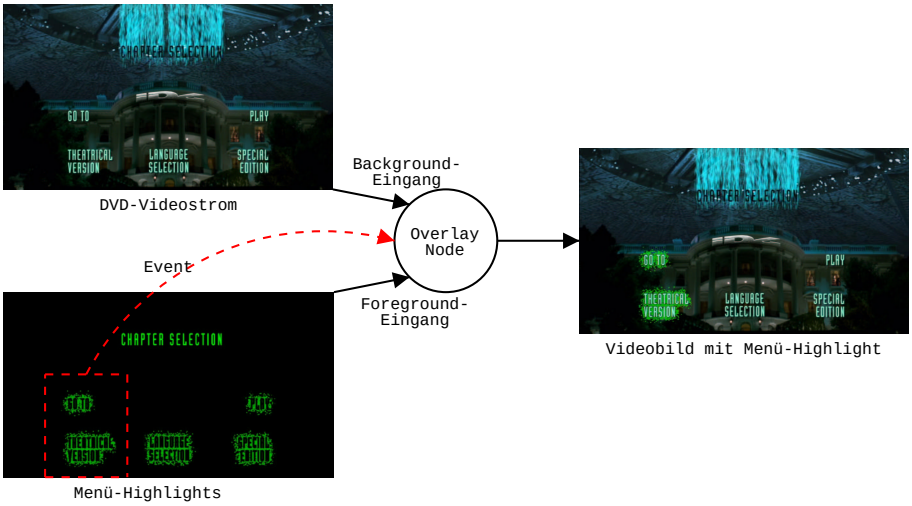

Abbildung 5.35: Der OverlayNode kann beliebige Grafiken auf einen Videostrom blenden. Bei einem DVD Menü blendet er Menüknöpfe, die vom SPUDecodeNode erzeugt wurden auf das Videobild. Die Menüknöpfe eines DVD-Menüs sind in einer Bitmap untergebracht und müssen extrahiert werden. Die Koordinaten und Größe des Rechtecks, das auf das Videobild geblendet werden soll, wird über das Event *overlay_segment_info* an den OverlayNode geschickt.

Menüknöpfe ersetzt werden, da sie auch gebraucht werden, wenn keine Daten mehr fließen, etwa bei einem Menü mit statischem Hintergrund. Da die Menüknopfbitmap alle Hervorhebungsbereiche der Knöpfe eines Menüs beinhaltet, darf nur ein bestimmter Bereich daraus auf das Videobild geblendet werden, der Knoten schickt deshalb verschieden Events weiter:

- **overlay_info** (*int x, int y, int width, int height)* “x,y” Enthält die obere, linke Ecke an der die Bitmap gezeichnet werden soll, “width” enthält die Breite und “height” die Höhe der Bitmap.
- **overlay_segment_info** (*int x, int y, int width, int height)* Diese Event wird bei Bitmaps mitgeschickt, die Menüknöpfe enthalten, dabei wird die Startposition (x,y) innerhalb der Bitmap und die Breite (width) und Höhe (height) des Knopfes mitgeschickt.

5.2.2.3 OverlayNode

Im Rahmen dieser Arbeit wurde der OverlayNode erweitert [65]. Zunächst hatte er die Fähigkeit eine Grafik über einen laufenden Videostrom zu blenden. Dabei mussten sowohl die Grafik als auch der Videostrom im RGB-Format vorliegen. Der Knoten wurde erweitert, so dass er neben dem RGB-Farbformat auch das YV12 Format unterstützt und um das Überblenden einer Grafik auf einen Videostrom

durchzuführen, wird eine assembleroptimierte Overlay-Routine verwendet (vgl. Abschnitt 5.2.2.1).

Der OverlayNode wird dazu verwendet Untertitel und Hervorhebungsbereiche für Menüknöpfe, die von dem SPUDecodeNode erzeugt werden, auf ein DVD-Videostrom zu blenden (vgl. Abschnitt 5.2.2.2). Der Knoten reagiert auf die Events *overlay_info* und *overlay_segment_info* (siehe Abschnitt 5.2.2.2). Es können somit reine Untertitel bzw. beliebige Bilder als auch Menüknöpfe angezeigt werden, die aus einer Bitmap extrahiert werden.

Ein spezielles Event, auf das der Knoten reagiert, ist das *still_frame* Event, das von dem DVDNavReadNode verschickt wird, wenn ein DVD-Menü kein laufendes Video hat, sondern einen statischen Hintergrund. Der OverlayNode speichert dann das letzte erhaltene Videobild in einem Zwischenpuffer, damit er bei einem Knopfwechsel im Menü, den Hervorhebungsbereich über diesen Zwischenpuffer blenden kann. Ohne Zwischenpufferung hätte der Knoten keine Hintergrundbilddaten mehr.

Der OverlayNode hat zwei Eingänge, einen “background” Eingang, an dem ein laufender Videostrom angeschlossen wird und die am “foreground” Eingang ankommenden Bilder werden über den laufenden Videostrom geblendet (vgl. Abbildung 5.35).

Die Fähigkeiten des OverlayNode hätten auch in den OSDManagerNode integriert werden können (vgl. Abschnitt 5.2.2.1). Da jedoch nicht alle mit dem NMM Framework entwickelten Anwendungen die Fähigkeiten des OSDManagerNode benötigen, wie z.B. das Anzeigen einer Message-Box oder das Verwalten von Zeichensätzen, sondern die Möglichkeiten des OverlayNode ausreichen, wurden die Knoten nicht zusammengefasst.

5.2.2.4 MPEGDemuxNode

MPEG Datenströme, die auf Video-DVDs vorhanden sind oder die zur Übertragung von digitalen Fernsehen verwendet werden, beinhalten Video, Audio und weitere Informationen (Untertitel). Dabei sind die unterschiedlichen Datenströme nicht getrennt auf dem Medium gespeichert, sondern sie werden gemultiplext, d.h. jeder einzelne Strom wird in kleine Teile (Pakete) gespalten. Diese Pakete werden abwechselnd in den Datenstrom geschrieben und mit Informationen wie Länge und Art des Pakets versehen, damit sie später beim Abspielen wieder getrennt werden können (Demultiplexing).

Die Aufgabe des Demultiplexers ist es, den gemultiplexten Datenstrom zu analysieren, und die einzelnen Ströme zu extrahieren. Der MPEGDemuxNode übernimmt diese Aufgabe. Er wurde schon für den Multimedia-Box Prototypen verwendet [65] und wird hier um einige Funktionen erweitert.

Der Eingabestrom, den der MPEGDemuxNode erwartet, muss im MPEG2 Containerformat (PS/PES) vorliegen [44]. Der Knoten kann die im Strom enthaltenen Einzelströme wie MPEG-Audio, AC3-Audio, MPEG-Video und SPU extrahieren und an die entsprechenden Ausgänge schicken. In einem Strom können meh-

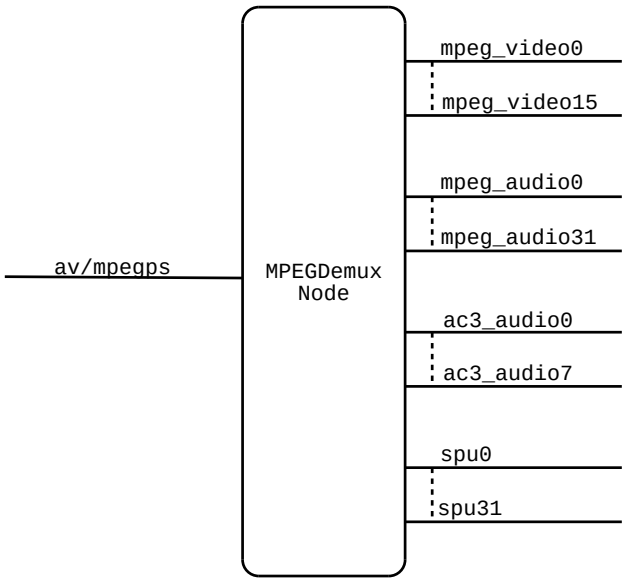

Abbildung 5.36: Am MPEGDemuxNode können alle Einzelströme eines MPEG Stroms gleichzeitig abgegriffen werden

rere Audio, Video und SPU Spuren enthalten sein. Der MPEGDemuxNode stellt deswegen für jeden Strom einen eigenen Ausgang zur Verfügung. Insgesamt hat er 88 Ausgänge. 16 für MPEG Video, 32 für SPU, 32 für MPEG Audio und 8 für AC3 Audio (Abbildung 5.36). Nützlich ist die parallele Bereitstellung zum Beispiel bei der Extraktion der Audiospuren einer Video-DVD. Es können parallel alle Audiospuren auf Platte geschrieben werden. Beim Abspielen einer Video-DVD kann zwischen verschiedenen Audio- und Untertitelspuren gewechselt werden. Hier ist es nützlich, dass man einen Ausgang so umschalten kann, dass er genau auf die ausgewählte Audio- oder Untertitelspur wechselt. Der MPEGDemuxNode kann per *set_dvd_mode* in einen speziellen Modus gewechselt werden, bei dem man mit den Events *set_video_stream*, *set_audio_stream* und *set_spu_stream* den jeweils ersten Ausgang so umschalten kann, damit er die gewählt Spur liefert (Abbildung 5.37).

5.2.2.5 MPEGTimeshiftingNode

Der MPEGTimeshiftingNode dient zur Playback-Kontrolle von Live-Quellen. Eine Live-Quelle kann z.B. eine über Satellit ausgestrahlte TV-Sendung oder eine Kamera, die für eine Videokonferenz benutzt wird, sein. Als Live-Quelle kann der in Abschnitt 5.2.1.2 vorgestellte DVBReadNode verwendet werden. Der MPEGTimeshiftingNode bietet die Möglichkeit, dass Aktionen wie Pause, Vor- und Zurückspulen auch auf Live-Quellen angewandt werden können. Natürlich lassen sich Live-Quellen nur bis zum Live-Bild vorspulen, bzw. nur bis zu einem gewissen Zeitpunkt zurückspulen.

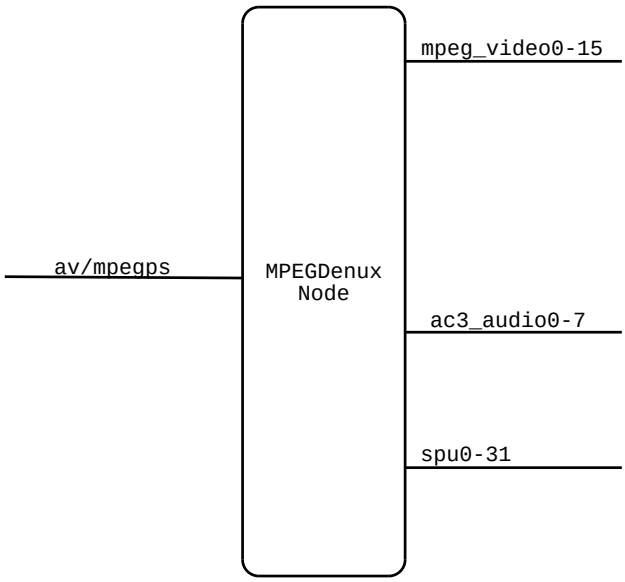

Abbildung 5.37: Ist der MPEGDemuxNode im DVD Modus kann nur noch ein Audio, Video oder SPU Strom abgegriffen werden. Die Audio, Video oder Untertitelspur kann aber während dem Demultiplexen gewechselt werden

Der MPEGTimeshiftingNode hat zwei Zustände. Der erste Zustand wird “Live-Modus” genannt, der zweite “Buffer-Modus”. Abbildung 5.38 zeigt die Zustände mit entsprechenden Zustandsübergängen.

Beim Starten des MPEGTimeshiftingNode befindet er sich im “Live-Modus”, d.h. eingehende Audio/Videodaten werden direkt zum Ausgang geleitet. Soll die Wiedergabe pausiert werden, so wechselt der Knoten in den “Buffer-Modus”. Eingehende Buffer werden jetzt auf Festplatte zwischengespeichert und es werden keine Daten mehr ausgegeben. Sie werden erst wieder ausgegeben, wenn die sie wiedergegeben werden sollen (Play) oder Vor- oder Zurückgespult werden soll. Das Spulen ist im “Buffer-Modus” möglich, da die eingehenden Daten während der Pause auf die Festplatte gesichert wurden.

Der “Live-Modus” wird wieder aktiv, wenn im “Buffer-Modus” bis zur Live-Position, d.h. bis zum Dateiende der aufgezeichneten Daten vorgespult wird oder wenn der “Buffer-Modus” explizit gestoppt wird.

Ein Besonderheit des “Live-Modus” ist das sogenannte “Instant Replay”, d.h. wird der “Live-Modus” gestoppt ohne dass der “Buffer-Modus” je aktiv war, also noch keine Daten auf der Festplatte liegen, die abgespielt werden könnten, dann werden einige Sekunden Audio/Video abgespielt, die im Hauptspeicher permanent zwischengespeichert sind. Beim Aktivieren des “Instant Replay” wird wieder in den “Buffer-Modus” gewechselt, damit die Daten wieder auf die Festplatte geschrieben werden können. Der MPEGTimeshiftingNode ist ein Knoten, der unver-

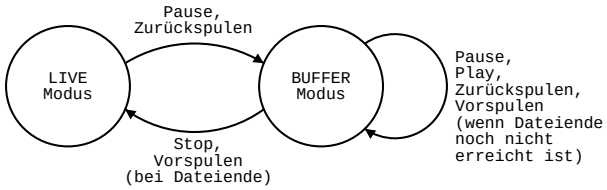

Abbildung 5.38: Zustände und Zustandsübergänge des MPEGTimeshiftingNode.

ändert aus [65] übernommen wurde.

5.2.2.6 MPEGVideoDecodeNode

Damit Videoströme platzsparend übertragen werden können, werden sie vorher komprimiert. Eine Möglichkeit Videos zu komprimieren ist die MPEG Kompression. MPEG komprimierte Videos findet man auf allen Video-DVDs, Video-CDs und in den digital übertragenen Fernsehprogrammen. Ein MPEG komprimiertes Video wird mit dem MPEGVideoDecodeNode dekodiert. Unterstützt werden sowohl MPEG1 als auch MPEG2 Videoströme.

Der MPEGVideoDecodeNode liefert die dekodierten Bilder an seinem Ausgang im YV12 Farbformat. Als Eingangsformat akzeptiert der Knoten “video/mpeg”. Einzelheiten über Bitrate, Framerate und Auflösung der Videobilder müssen hier nicht angegeben werden, sie werden vom Knoten selbst aus dem Videostrom extrahiert. Danach wird das Event *resolution_changed*, das die Breite und Höhe eines Bildes und das Seitenverhältnis enthält, verschickt. Die Framerate wird dadurch erzielt, dass jeder ausgehende Buffer, der das dekodierte Videobild enthält, zusätzlich mit einem Zeitstempel versehen wird, der dann von einem Senkeknoten, der das Videobild anzeigt, ausgewertet wird.

Der MPEGVideoDecodeNode benutzt die libmpeg2-Library [6] zum dekodieren des MPEG-Stroms.

Sobald Audio und Video parallel abgespielt werden, muss darauf geachtet werden, dass die Daten zusammenpassen, d.h. Audio und Video müssen miteinander synchronisiert werden. Verschiedene Strategien, wie einzelne Datenströme insbesondere in der NMM-Architektur synchronisiert werden, kann in [74] nachgelesen werden. Der MPEGVideoDecodeNode ist ein Knoten, der unverändert aus [65] übernommen wurde.

5.2.2.7 RGBtoYV12ConverterNode

Der RGBtoYV12ConverterNode wandelt Bilder, die im RGB Format vorliegen, in das YV12 Format. Grafikkarten können meist nur Videos, die in einem YUV Format vorliegen, hardwarebeschleunigt skalieren. Liegen die Bilder nur im RGB Farbformat vor, müssen sie zuerst nach YV12 konvertiert werden. Weitere Informationen über den RGBtoYV12ConverterNode befinden sich in [65].

5.2.2.8 MPEGAudioEncodeNode

Er enkodiert unkomprimierte Audiodaten wie sie auf einer Audio-CD zu finden sind in das MPEG1 layer 3 (MP3) Format. Der Knoten akzeptiert Audiodaten mit einer Samplingfrequenz von 11025Hz bis 48Khz mit ein oder zwei Kanälen. Enkodiert wird mit Hilfe der Lame-Library [61]. Es können sowohl verschiedene Bitraten als auch unterschiedliche Qualitäten ausgewählt werden. Die Qualität beeinflusst dabei direkt die Zeit, die der Knoten braucht um die unkomprimierten Audiodaten zu komprimieren. Der MPEGAudioEncodeNode wurde unverändert aus [65] übernommen.

5.2.2.9 MPEGAudioDecodeNode

Mit dem MPEGAudioDecodeNode können MPEG-Audio kodierte Daten dekodiert werden. Unterstützt werden die Formate MPEG1 layer 1 bis 3. Um den Ausgang eines Knoten mit dem Eingang des MPEGAudioDecodeNode zu verbinden, muss nur der Haupttyp “audio” und der Untertyp “mpeg” sein. Zusätzliche Parameter müssen nicht angegeben werden. Die Bitrate, Anzahl Kanäle und Frequenz werden vom Knoten aus dem Audiostrom extrahiert. Wenn der Knoten das komplette Format erkannt hat, wird das Event *set_output_format* gesendet, das die Bitrate, Kanäle und Frequenz enthält.

Das Ausgangsformat hat den Haupttyp “audio” und den Untertyp “raw”. Der Eingang des PlaybackNode kann direkt mit Ausgang des MPEGAudioDecodeNode verbunden werden.

Der Knoten basiert auf der Mad-Library [70], die alle oben erwähnten MPEG-Layer dekodieren kann. Der MPEGAudioDecodeNode wurde unverändert aus [65] übernommen.

5.2.2.10 AC3DecodeNode

In Abschnitt 5.2.3.1 wird der PlaybackNode und seine Fähigkeit, AC3 Daten zu verarbeiten, erklärt. Soundkarten die keine AC3-Unterstützung haben können mit Hilfe des AC3DecodeNode trotzdem AC3-Ströme abspielen.

AC3 Audioströme können bis zu 6 Kanäle beinhalten. AC3 Daten befinden sich auf fast allen Video-DVDs. Der AC3DecodeNode konvertiert einen eingehenden AC3-Strom in einen unkomprimierten Stereo PCM Strom, der mit jeder Soundkarte abgespielt werden kann. Der Knoten greift zum Dekodieren des AC3-Stroms auf die Library liba52 [5] zurück. Der AC3DecodeNode wurde in [65] entwickelt und wird für diese Arbeit unverändert übernommen.

5.2.3 Senkeknoten

5.2.3.1 PlaybackNode

Der PlaybackNode ist für die Soundausgabe verantwortlich. Er ist ein Senke, hat also nur einen Eingang. Das Eingangsformat hängt von den Fähigkeiten der Soundkarte ab. Das Format umfasst dabei Sampling-Frequenz, Samplebreite und Anzahl der Kanäle. Übliche Audioströme sind entweder Mono oder Stereo, haben eine Samplebreite von 8 oder 16 Bit und eine Sampling-Frequenz von bis zu 48 KHz.

Die Implementierung des PlaybackNode im Multimedia-Box Prototypen unterstützt alle oben genannten Formate. Er wurde in dieser Arbeit erweitert, so dass im Vergleich zu der Version in [65] auch Audioströme im AC3 Format unterstützt werden.

Audioströme im AC3-Format befinden sich auf vielen Video-DVDs. In einem AC3-Audiostrom können bis zu 6 Kanäle kodiert werden. Zwei Frontkanäle, zwei Rückkanäle, ein Centerkanal und ein Kanal für sehr tiefe Frequenzen (Subwoofer). Wird dieses Format von der Soundkarte unterstützt kann der encodierte AC3-Strom zur Soundkarte geschickt werden, wo er entweder in 6 Kanäle dekodiert wird oder über eine digital Verbindung zur einem externen Receiver übertragen wird und dort dekodiert wird.

Die Übertragung zu einem externen Receiver geschieht wiederum über eine digitale Schnittstelle. *S/PDIF* (Sony/Phillips Digital InterFace) [35] ist die digitale Schnittstelle, die im Consumerbereich verwendet wird. Soundkarten mit einer S/PDIF Schnittstelle können unkodierte Audiodaten direkt an den digitalen Ausgang schicken. Digitale Stereo-Audiodaten können von jedem Receiver mit einer S/PDIF Schnittstelle verarbeitet werden. AC3-Audioströme können von Receivern mit AC3-Dekoder verarbeitet werden. Soundkarten, die AC3-Audiodaten nicht dekodieren und eine S/PDIF Schnittstelle besitzen, können jedoch AC3-Daten über diese Schnittstelle an einen externen Receiver weiterleiten (engl. AC3 passthrough).

Bevor die AC3-Daten von der Soundkarte weitergeleitet werden können, müssen sie entsprechend aufbereitet werden. AC3-Daten können nur Frameweise und mit einem speziellen Header von der Soundkarte (Bsp. Soundblaster Live) verarbeitet werden. Eine Aufgabe des PlaybackNodes ist es, AC3-Audiodaten zu sammeln bis ein Frame vollständig ist oder wenn ein einkommender Buffer mehrere Frames hat, diese zu trennen. In [7] wird der Aufbau eines AC3-Frames detailliert erklärt. Zur Entscheidung, ob ein Buffer ein vollständiges Frame oder mehrere Frames besitzt, werden die ersten 5 Bytes eines AC3 Frames (Abbildung 5.39) herangezogen.

Das erste Word ist das Synchronisationswort. Es hat einen Wert von 0xB77. Es ist sichergestellt, dass dieser Wert im ganzen AC3 Frame nicht wiedervorkommt. Den Anfang eines AC3 Frames kann man also durch Suchen des Synchronisationswortes lokalisieren. Danach folgt eine 16 Bit Checksumme, die zur Fehlererkennung benutzt werden kann. Nachfolgende 8 Bits enthalten die Samplingfrequenz und die Framegröße. Die oberen 2 Bits enthalten den Code der Samplingfrequenz,

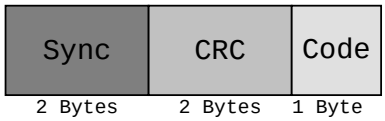

Abbildung 5.39: Die ersten fünf Bytes eines AC3 Frames.

Code	Samplefrequenz (in KHz)
00	48
01	44,1
10	32
11	reserviert

Tabelle 5.1: Samplefrequenz-Codes

die in Tabelle 5.1 aufgeführt sind. Die unteren 6 Bits enthalten kodiert die Bitrate des AC3 Stroms und die Größe des Frames. In Tabelle A.4 können diese Werte nachgeschlagen werden.

Bevor das Frame zur Soundkarte geschickt werden kann, muss es mit einem Header versehen werden. Abbildung 5.40 zeigt dessen Aufbau. Die ersten 4 Bytes haben einen festen Wert von 0x72F81F4E und dienen zur Synchronisation mit der Soundkarte. Danach folgt ein fester Wert von 0x0100, wenn die Größe des AC3 Frames > 0 ist, ansonsten steht dort 0x0000. Die Länge des AC3 Frames muss in den oberen 13 Bits des nächsten Wortes untergebracht werden.

5.2.3.2 XDisplayNode

Um Videobilder und grafisches Benutzerinterface darzustellen, wird ein Knoten zur Anzeige von Grafiken benötigt. Dieser Knoten wird XDisplayNode genannt. Er kann ein X Window erstellen, um dort eingehende Bilder darzustellen. Dabei hängen seine Fähigkeiten und Eingabeformate von der Grafikhardware und deren XFree [84] Treiber ab. In Abschnitt 4.2.4 wurden alle Fähigkeiten aufgelistet, die für die Anzeige von Videoströmen und digitalen Fernsehprogrammen benötigt werden.

Der XDisplayNode reagiert auf das Event *resolution_changed*. Das bedeutet,

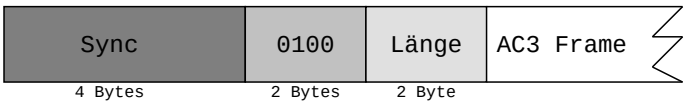

Abbildung 5.40: Aufbau des Headers, der vor jedes AC3 Frame platziert werden muss, bevor es zur Soundkarte geschickt werden kann.

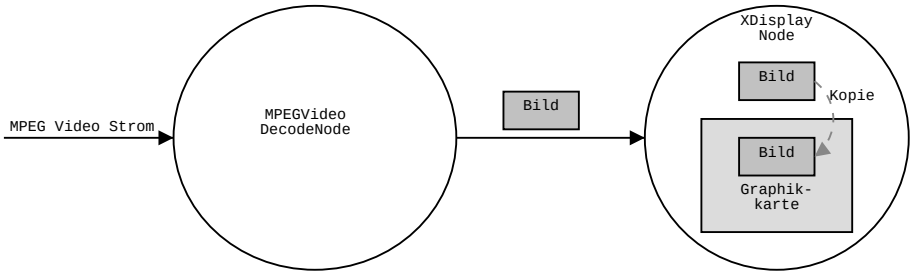

Abbildung 5.41: Der MPEGVideoDecodeNode schickt ein dekodiertes Bild zum XDisplayNode, dort muss es nochmal in den Speicher der Grafikkarte kopiert werden

immer dann, wenn sich das Bildformat ändert, reagiert der Knoten darauf und vergrößert oder verkleinert sein X-Ausgabefenster.

Neben dem von NMM bereitgestellten Buffer-Manager kann der XDisplayNode auch einen Shared Memory Buffer-Manager benutzen.

Der Buffer-Manager eines NMM Knoten fordert seine Speicherbereiche vom Betriebssystem an, d.h. er verwaltet nur den Hauptspeicher (vgl. Abschnitt 3.4). Oft ist es aber sinnvoll "externe" Speicherbereiche zu verwalten wie etwa den Speicher einer Grafikkarte, um die Performance zu verbessern. Diese Aufgabe übernimmt der Shared Memory Buffer-Manager. Dieser Buffer-Manager kann jegliche Art von Speicherbereichen verwalten, da er eine Schnittstelle besitzt, mit der man Speicherbereiche hinzufügen kann. Mit der Methode

```
addSharedMemory( void* memptr, unsigned int size )
```

kann man einen Speicherbereich hinzufügen. `memptr` ist dabei ein Zeiger auf den Speicher, der hinzugefügt werden soll und `size` dessen Größe. Es können beliebig viele Speicherbereiche hinzugefügt werden. Besitzt ein Knoten einen Shared Memory Manager, so bekommt er bei einer Anfrage mit `getNewBuffer` einen Buffer, der einen Speicherbereich aus dem Speicherpool enthält. Sind keine Buffer im Pool verfügbar, so blockiert der Aufruf so lange, bis ein Buffer frei wird.

Die XServer neuerer Grafikkarten besitzen die MIT Shared Memory Extension [43]. Diese Extension erlaubt es, Bilddaten in einem gemeinsam benutzten Speicherbereich abzulegen und eine Schnittstelle, mit der man ohne Xlib-Interprozesskommunikation diese Bilddaten anzeigen kann. Das bedeutet, dass die Bilddaten entweder direkt im Speicher der Grafikkarte angelegt werden oder in einem Speicherbereich, auf den die Grafikkartenhardware direkt zugreifen kann. Diese Besonderheit wird von allen AGP-Grafikkarten unterstützt.

Ein Videobild, das vom MPEGVideoDecodeNode dekodiert wurde, braucht ca. 1 MB Speicherplatz (DVD Qualität). Um eine flüssige Bildwiedergabe zu erhalten, müssen pro Sekunde 25 oder 29,97 Bilder angezeigt werden, d.h. der XDisplayNode muss pro Sekunde 25 MB Daten zur Grafikarte schicken (Abbildung 5.41). Der Shared Memory Buffer-Manager erlaubt, dass der MPEGVideoDecodeNode

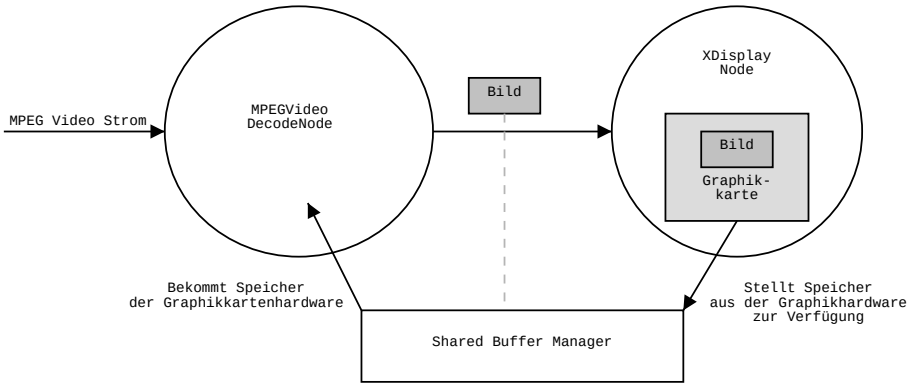

Abbildung 5.42: Der MPEGDecodeNode schickt ein dekodiertes Bild zum XDisplayNode, dabei liegt das dekodierte Bild bereits in einem Speicherbereich, auf den die Grafikkartenhardware zugreifen kann. Im XDisplayNode braucht das Bild nicht kopiert zu werden, es reicht, der Grafikkarte zu sagen, dass das Bild angezeigt werden soll

die Bilder direkt in dem von der Grafikkarte bereitgestellten Speicherbereich dekodiert. Somit entfällt die Kopie der Daten im XDisplayNode (Abbildung 5.42).

Der Konstruktor des XDisplayNode erwartet die Anzahl der gemeinsam benutzten Speicherseiten. Dabei hat eine Speicherseite standardmäßig die Größe eine PAL-Bildes im YV12 Format (1.036.800 Bytes), wird keine Seitenanzahl angegeben wird kein shared Memory benutzt. Die festen Seitengröße wurde gewählt, da die eingesetzte Grafikkarte scheinbar nicht alle Größen akzeptiert. In wie weit das ein Fehler im benutzten Grafiktreiber ist, kann zu diesem Zeitpunkt nicht eingeschätzt werden. Eine spätere Implementierung könnte jedoch ein Interface zum Setzen der Seitengröße anbieten.

Damit ein Knoten weiß, dass er einen anderen Speichermanager benutzen soll, muss man ihn mit `setBufferManager` setzen. Damit der MPEGVideoDecodeNode wie in Abbildung 5.42 gezeigt, den Speicher der Grafikkarte benutzt, sind folgende Anweisungen nötig:

```
MPEGVideoDecodeNode* mpg2yuv;
/* erstelle Videodekoder, der keine Buffer verwirft
 * und eine Queuegröße von 200 hat
 */
mpg2yuv = new MPEGVideoDecodeNode("MPEG",
          StreamQueue::MODE_SUSPEND, 200);

XDisplayNode* display;
/* erstelle XDisplayNode, der keine Buffer verwirft,
 * eine Queuegröße von 1 hat und
 * 6 Speicherseiten hat
 */
```

```
display = new XDisplayNode("Display",
          StreamQueue::MODE_SUSPEND, 1, 6 );
```

Nachdem der MPEGVideoDecodeNode und XDisplayNode mit shared Memory erstellt wurden, muss der Shared Memory Buffermanager mit

```
display->requestBufferManager()
```

beim XDisplayNode angefordert und anschließend beim MPEGVideoDecodeNode gesetzt werden.

```
mpg2yuv->setBufferManager(
  display->requestBufferManager() );
```

Der XDisplayNode wurde aus [65] übernommen und durch den shared Memory Buffer-Manager erweitert.

5.3 Benutzereingaben und Ausgaben

Es gibt unterschiedliche Eingabegeräte wie zum Beispiel Tastatur oder Fernbedienung, die zur Steuerung der Multimedia-Box verwendet werden können. Das Betriebssystem behandelt alle Eingabegeräte unterschiedlich. Wenn zum Beispiel eine Taste auf der Tastatur gedrückt wird, liefert ein Systemaufruf wie `getchar()` die gedrückte Taste. Wenn eine Taste auf der Fernbedienung gedrückt wird, muss wiederum ein anderer Systemaufruf ausgeführt werden. Als Erweiterung könnten auch andere Eingabemöglichkeiten realisiert werden. Beispielsweise könnte die Multimedia-Box über das Netzwerk oder über Sprache gesteuert werden.

Sinnvoll ist deswegen eine gemeinsame Schnittstelle, die alle Eingabegeräte abdeckt. Diese Schnittstelle wird in Form von “Producer” Objekten realisiert. Producer enthalten Funktionen, die Eingaben der verschiedenen Geräte abfangen und diese dem Anwenderprogrammierer in Form von Events zurückliefern. Ein Producer kann an einen Eventdispatcher (Abschnitt 3.3) angeschlossen werden, der automatisch Handlerfunktionen für die Tastendrücke aufruft.

In den nachfolgenden Abschnitten werden zwei Producer erklärt, die schon in [65] entstanden sind.

5.3.1 Infrarotfernbedienung - LircProducer

Der LircProducer empfängt seine Tastendrücke von einer Fernbedienung, dabei dient der LIRC Daemon [16] als Schnittstelle zwischen dem Empfängermodul und dem Producer. Die Events, die der Producer generiert, werden durch eine Konfigurationsdatei (Anhang A.5) eingestellt. In dieser Konfigurationsdatei werden den Tasten der Fernbedienung verschiedene Namen zugewiesen. Der LircProducer verpackt diese Namen in ein NMM-Event und leitet es an einen Event-Dispatcher weiter. Ein kurzer Code-Ausschnitt zeigt, wie ein LircProducers initialisiert und gestartet wird:

- Zuerst muss ein LircProducer erstellt werden:

```
LircProducer = new LircEvents();
```

- Einlesen der Konfigurationsdatei

  ```
  LircProducer->readConfig("remote.conf");
  ```

- Verbinden des Producers mit einem Dispatcher

  ```
  LircProducer->connectTo((EventDispatcher*) dispatcher);
  ```

- Starten des Producers

  ```
  LircProducer->start();
  ```

5.3.2 XProducer

Der XProducer empfängt Tastendrücke von der Tastatur, wenn eine Anwendung in einem X-Fenster läuft. Er wandelt dabei die Tastenevents, die er vom X Window System erhält entsprechend einer Konfigurationsdatei um. Die Konfigurationsdatei ist ein XML File und hat folgenden Aufbau:

```
<keyboard>
  <event key="a">KEY_a</event>
  <event key="b">KEY_b</event>
  ...
  <event key="UP">KEY_Up</event>
  <event key="DOWN">KEY_Down</event>
  ...
</keyboard>
```

Mit dieser Konfiguration wird der Producer angewiesen, immer wenn er vom X Window System die Taste “a” empfängt, ein Event “KEY_a” wegzuschicken. Hier eine kurze Beschreibung, welche Schritte nötig sind, um einen XProducer zu initialisieren:

- Zuerst muss ein XProducer erstellt werden:

  ```
  Xevents* xproducer = new Xevents();
  ```

- Dann muss dem Producer eine Display ID mitgeteilt werden, die man vom XDisplayNode erhält:

  ```
  xproducer->registerDisplay(
       ((XDisplayNode*) display)->getID());
  ```

- Einlesen der XML Konfigurationsdatei

  ```
  xproducer->readConfig("configuration.xml");
  ```

- Verbinden des Producers mit einem Dispatcher

```
xproducer->connectTo((EventDispatcher*) dispatcher);
```

- Starten des Producers
  ```
  xproducer->start();
  ```

5.3.3 Ansteuerung des LC-Displays

Die Klasse `LCDWriter` bietet Funktionen, um das LC-Display (Abschnitt 4.3.2) anzusteuern. Sie kann einfache Texte, Fortschrittsbalken, Datum und Zeit anzeigen. Damit Informationen automatisch an das Display geleitet werden, ist es nützlich einen Event-Handler anzulegen, der auf bestimmte Events (zum Beispiel Setzen des Dateinamens) reagiert und auf dem Display anzeigt.

Die Klasse `LCDWriter` wurde im Rahmen des Fortgeschrittenen-Praktikums entwickelt [65]. Diese Klasse bietet folgende Schnittstelle an:

Result init() Initialisiert das LC-Display, d.h. es wird geprüft, ob der Treiber geladen ist und das Display korrekt funktioniert. Konnte das Display nicht initialisiert werden, wird `FAILURE` zurückgegeben.

int clear() Löscht den gesamten Text, der momentan auf dem Display angezeigt wird.

int setText (int line, const char* text) Schreibt den angegebenen Text auf das Display, dabei kann eine Zeilennummer angegeben werden.

int setDateTime (int line) Schreibt das aktuelle Datum und die aktuelle Uhrzeit auf das Display, “line” gibt die Zeilennummer an, in der der Text dargestellt werden soll.

int setChar(int x, int y, char c) Stellt das einzelne Zeichen “c” an der angegebenen Position “(x,y)” dar.

int clearLine(int line) Löscht die angegebene Zeile.

void setProgressBar(int line, int percent) Zeichnet in der angegebenen Zeile einen Fortschrittsbalken. Die Länge des Balkens wird in Prozent angegeben, d.h. bei einem Wert von 100 füllt der Balken die ganze Zeile aus.

Result receiveStartTrack(string& text) Die Methode wurde für die in Kapitel 6 vorgestellte Applikation implementiert. Sie gibt den angegeben Text auf dem LCD aus. Handelt es sich bei dem Text um einen Dateinamen mit voller Pfadangabe, so wird der führende Pfad abgeschnitten und nur der reine Dateinamen ausgegeben.

Folgendes Beispiel gibt einen Text auf dem LCD aus:

```
LCDWriter lcdWriter;
lcdWriter.init();
lcdWriter.setText(0, "Hallo␣Welt");
```

Kapitel 6

Die Multimedia-Box Anwendung

In diesem Kapitel wird eine Anwendung beschrieben, mit der ein Benutzer u.a. CDs und DVDs abspielen kann, Fernsehen schauen und Fernsehsendungen aufzeichnen kann und verschiedene Multimedia-Dateien abspielen kann. Diese Anwendung wird Multimedia-Box genannt.

Die Multimedia-BoxAnwendung kann mit einer Fernbedienung bedient werden. Sie hat eine hierarchische Menüstruktur, in dem der Benutzer navigieren kann. Einzelne Funktionalitäten wie CD-Spielen usw. werden durch grafische Symbole repräsentiert, die der Benutzer einfach anwählen und starten kann. Während die Funktion ausgeführt wird hat der Benutzer aber immernoch Interaktionsmöglichkeiten (Multitasking).

Das Kapitel gliedert sich in zwei Teile, wobei der erste Teil (Abschnitt 6.1 bis 6.1.10) alle Funktionalitäten der Multimedia-Box Anwendung beschreibt. Im einzelnen werden folgende Funktionalitäten beschrieben:

- DVD-Spieler mit grafischem Navigationsmenü. Damit können Video-DVDs abgespielt werden. Dabei können in einem Menü Sprache, Untertitel und verschiedenes Bonusmaterial ausgewählt werden.
- CD-Spieler, der Audio-CDs abspielen kann. Die einzelnen Lieder einer CD können über eine Liste ausgewählt werden, die u.a. auch die genaue Bezeichnung und den Interpret jedes einzelnen Titels der CD enthält.
- CD-Grabber, mit dem Audio-CDs platzsparend auf die Festplatte archiviert werden können.
- TV-Viewer, mit dem digitale Fernseheprogramme wiedergegeben werden können und mit dem Timeshifting-Funktionen durchgeführt werden können.
- TV-Timer, mit dem Fernsehprogramme programmiert und zeitgesteuert aufgenommen werden können.
- MP3-Spieler, der MP3-Audiodateien und MP3-Playlisten abspielen kann. Mit ihm können auch die archivierten Titel des CD-Grabbers abgespielt werden.

- Playlist, die es dem Anwender ermöglicht verschiedene Dateien in unterschiedlichen Formaten, wie z.B. Audio- und Video-Dateien in eine Playlist aufzunehmen und abzuspielen. Die Einträge einer Playlist können gespeichert, verschoben oder gelöscht werden.

- Taskmanager, in dem alle laufenden Funktionen aufgeführt sind. Der Taskmanager erlaubt auch das hin- und herschalten zwischen verschiedenen aktiven Funktionen, die auch gestoppt werden können.

- Konfigurationsmenü, in dem einige der oben aufgeführten Anwendungen konfiguriert und ihr Verhalten verändert werden können.

Der zweite Teile (Abschnitt 6.2 bis 6.3.11) beschreibt die Architektur der Anwendung, sowie die Implementierung der einzelnen Funktionalitäten. Die Implementierung stützt sich im wesentlichen auf die in Abschnitt 5.1 entwickelte grafische Oberfläche und die Verwendung von NMM-Knoten (vgl. Abschnitt 5.2). Für jede Funktionalität wird der entsprechende NMM-Flussgraph vorgestellt.

Alle Funktionalitäten werden durch eine XML-Datei konfiguriert und durch einen XML-Parser in die Hauptanwendung eingebunden. Durch Verwendung des State-Design Patterns (Abschnitt 6.2) werden alle Anwendungen als einzelne Zustände betrachtet. Zustände, also einzelne Funktionalitäten können parallel ausgeführt werden. Zum Beispiel kann eine CD nach MP3 konvertiert werden während das Fernsehprogramm wiedergegeben oder aufgenommen wird. Beim parallelen Abspielen wird darauf geachtet, dass nicht zwei Anwendungen gleichzeitig Ressourcen benutzen, die nur einmal vorhanden sind, wie Bildschirm, Soundkarte oder DVD-Laufwerk. Auch Navigationsmenüs sind als Zustände implementiert und können in einer XML-Datei konfiguriert werden.

Das Aussehen der Anwendung kann allein durch die Änderung dieser Datei bestimmt werden. Durch das Einbinden verschiedener Grafiken können sogenannte *Skins* realisiert werden. In der XML Datei wird beschrieben, welche Grafiken für die Menüknöpfe verwendet werden sollen und welche Aktion ausgeführt werden soll, wenn ein Knopf gedrückt wird. Die Menüs sind hierarchisch aufgebaut. Ein Menü kann mehrere Untermenüs haben. Auch das Verhalten der Zustände ist in der XML Datei konfigurierbar. Beispielsweise kann angegeben werden, ob der Zustand einen PlaybackNode oder einen Hintergrund benötigt, um Audio oder Videodaten abzuspielen. Dementsprechend können zwei Zustände, die diese “Ressourcen” zur gleichen Zeit benötigen, nicht parallel ausgeführt werden.

XML Dokumente werden für fast alle Konfigurationszwecke verwendet. Ausnahme bilden hier die Konfiguration von LIRC und der DVB API. Sie werden durch ihre eigenen Konfigurationsdateien angepasst. Auch Verzeichnisse, in denen die MMBox-Anwendung Daten ablegt, werden in einem Konfigurationsfile gesichert. Aufbau dieser “.mmboxrc” Datei kann im Anhang A.7.1 nachgeschlagen werden.

6.1 Bedienung der Multimedia-Box Anwendung

Die MMBox-Applikation präsentiert zu Beginn ein Hauptmenü, in dem man navigieren, in Untermenüs wechseln oder einzelne Funktionen starten kann. Das Menü ist hierarchisch aufgebaut. Navigiert wird mit einer Fernbedienung, wobei auch eine Tastatur benutzen werden kann. Da die Tastenbelegung von Tastatur und Fernbedienung frei eingestellt werden kann, werden folgende Bezeichnungen benutzt:

- Navigationstasten (Rechts, Links, Hoch, Runter)
- Play
- Stop
- Pause
- FF (wird meist zum Vorwärtsspulen benutzt)
- REW (wird meist zum Rückwärtsspulen benutzt)
- Back
- Funktion 1
- Funktion 2
- OSD
- Exit (Beendet aktuellen Zustand)
- Aus (Beendet die Anwendung)

Die Standardbelegung für PC-Tastatur und Panasonic-Fernbedienung kann in Anhang A.8 nachgeschlagen werden. Abschnitt 5.3.2 beschreibt, wie man eine PC-Tastatur konfigurieren kann, in Anhang A.5 wird die Konfiguration einer Fernbedienung beschrieben.

6.1.1 Haupt- und Untermenü

Das Hauptmenü gibt einen Überblick über einige verfügbare Zustände. Abbildung 6.1 zeigt ein solches Hauptmenü. Innerhalb von Menüs kann mit den Navigationstasten zwischen Menüknöpfen gewechselt werden. Die Play-Taste wechselt in ein Untermenü oder startet die mit dem Menüpunkt verbundene Anwendung. Menüs können beliebig viele Untermenüs haben. Ein Druck auf die Back-Taste bewirkt immer, dass in das vorherige Menü zurückgesprungen wird. Beenden wird ein Zustand durch Drücken von Exit.

Knöpfe und Hintergrund sind Bitmapgrafiken, die selbst entworfen werden können oder die man auf verschiedenen Internetseiten finden kann. Das Hintergrundbild von Abbildung 6.1 kann beispielweise in [18] gefunden werden. Hintergrundgrafiken müssen 720x576 Pixel groß sein, die Größe der Knöpfe kann variieren, sollten aber die Größe des Hintergrundbildes nicht überschreiten.

Abbildung 6.1: Hauptmenü der Multimedia Box Applikation mit dem “Neon3” Skin. Einzelne Menüknöpfe ermöglichen den Zugriff auf Untermenüs und die Anwendungen. Der aktuell selektierte Knopf ist hervorgehoben.

6.1.2 DVD-Spieler

Startet man den DVD-Player und eine Video-DVD befindet sich im Laufwerk, dann wird die Video-DVD direkt abgespielt, ansonsten wird eine Fehlermeldung ausgegeben, die man mit der Play-Taste bestätigen kann und es wird ins Hauptmenü gewechselt.

Wurde die Video-DVD als solche erkannt, wird sie abgespielt. Je nach Inhalt Video-DVD präsentiert sich dann ein Menü, in dem man Sprache, Untertitel und einzelne Szenen auswählen kann (Abbildung 6.2). Die Menüstruktur und die Gestalt der Knöpfe bei Video-DVDs kann sehr stark variieren. Alle DVD-Menüs lassen sich mit 5 Tasten steuern: 4 Navigationstasten zum Auswählen der Knöpfe und die Play-Taste, um die mit dem Menüknopf verbundenen Aktion wie Tonwahl, Szenenauswahl usw. auszulösen (vgl. Abschnitt 5.2.1.5). Während der Wiedergabe kann man mit der Pause-Taste die Wiedergabe anhalten. Durch nochmaliges Drücken der Taste wird die Wiedergabe fortgesetzt. Vor- und Zurückspulen ist momentan noch nicht möglich. Es kann jedoch im laufenden Film mit der FF-Taste ein Kapitel vorgesprungen und mit der REW-Taste ein Kapitel zurückgesprungen werden.

Abschließend noch einige Bemerkungen zur Stabilität des DVD-Spielers. Die Wiedergabe des Hauptfilms einer Video-DVD ist im Allgemeinen problemlos möglich. Jedoch bereiten einige DVD-Menüs noch Schwierigkeiten. So kann es vorkommen, dass zwar ein DVD-Menü erscheint, jedoch die Hervorhebung der Knöpfe fehlen. Dies tritt meist bei Menüs auf, die keine Hintergrundmusik haben und kein laufendes Video im Hintergrund. Solche “Still frame” Menüs müssen beson-

Abbildung 6.2: Ein Menü einer Video-DVD. Hier können beispielsweise die verschiedene Kapitel der DVD ausgewählt werden.

ders behandelt werden, jedoch werden sie manchmal nicht richtig als solche erkannt. Lösung dieses Problems könnte eine neuere Version der verwendeten Library libdvdnav [48] sein, die die Anwendung auch direkt beendet, wenn sie Fehler nicht ausreichend abfängt.

6.1.3 CD-Spieler

Eine Audio-CD kann abgespielt werden, indem man den CD-State startet. Durch Drücken des CD-Knopfes wird zunächst ein neues Untermenü präsentiert, indem ausgewählt werden muss, ob die CD abgespielt werden soll oder ob sie als MP3 auf Platte archiviert werden soll. In beiden Fällen wird zunächst geprüft, ob es sich bei der eingelegten CD um eine gültige Audio-CD handelt und eine entsprechende Fehlermeldung ausgegeben, wenn das Medium nicht gelesen werden konnte. Konnte die Audio-CD gelesen werden, erscheinen Interpret und Titel der einzelnen Tracks in einem Auswahlfenster. Interpret und Titel werden dabei via CDDB [77] Datenbank übers Internet abgerufen. Existiert keine Internetverbindung oder ist die CDDB Option deaktiviert, so wird der Interpret auf “Unknown” gesetzt und die Titel mit “Track01” usw. durchnummeriert.

Mit den Navigationstasten kann zwischen einzelnen Titeln selektiert werden. Ein Druck auf die Play-Taste spielt den selektierten Titel. Die FF-Taste und REW-Taste können zum Vor- und Zurückspulen innerhalb des Titels benutzt werden. Spieldauer und Fortschrittsanzeige werden in zwei weiteren Fenstern angezeigt (Abbildung 6.3).

Abbildung 6.3: Titel und Tracknamen einer CD werden via Internet abgefragt und in einer Auswahlbox präsentiert. Progressbar und Zeitanzeige geben Feedback über den aktuellen Status des wiedergegebenen Tracks.

6.1.4 CD-Grabber

Im Untermenü des CD-Punktes existiert der Unterpunkt “Grab”, mit dem eine Audio-CD ausgelesen und als MP3 archiviert werden kann. Dieser CD-Grabber präsentiert sich wie der CD-Spieler. Unterschied ist hier, dass die einzelnen Titel nicht abgespielt werden, sondern als MP3 auf Festplatte geschrieben werden. Dabei wird für jede CD ein eigenes Verzeichnis angelegt, das sich wiederum in einem speziellen Verzeichnis befindet, in dem die MMBox-Applikation alle Daten archiviert und wieder auslesen kann. Dieses Verzeichnis wird “Rootpath” genannt, dessen Wert in dem MMBox Konfigurationsfile (Anhang A.7.1) gespeichert ist. Im Unterverzeichnis “playlist” wird eine M3U-Datei generiert, die die Titel der archivierten Audio-CD in der richtigen Reihenfolge enthält. M3U-Playlistdateien können u.a. vom MP3-Player abgespielt werden. Im CD-Grabber kann nicht zwischen einzelnen Titeln navigiert werden. Der Auswahlbalken zeigt an, welcher Titel gerade archiviert wird. Der Fortschrittsbalken zeigt, wie weit der aktuelle Titel schon gesichert wurde.

Sind alle Titel der Audio-CD archiviert worden, wird die Auswahlliste um den Eintrag “Grab Finished” erweitert und durch den Auswahlbalken hervorgehoben.

Abbildung 6.4: Mit dem TV-Viewer kann mit Hilfe einer digitalen TV Karte das Fernsehprogramm wiedergegeben werden.

6.1.5 TV-Viewer

Der TV-Viewer (Abbildung 6.4) ersetzt einen herkömmlichen Satellitenreceiver oder Kabelfernsehen und erweitert sie zusätzlich um Timeshifting-Funktionen.

Sobald man in die TV-Viewer Anwendung wechselt, wird das aktuell eingestellte Fernsehprogramm wiedergegeben. Mit den Navigationstasten “hoch” und “runter” kann zwischen einzelnen Fernsehkanälen gewechselt werden. Die Programmliste und die Frequenzen der einzelnen Sender wird in einer Konfigurationsdatei abgelegt, die dem VDR entnommen ist (vgl. Anhang A.4).

Die Timeshifting-Funktionalität erlaubt es, während der Live-Wiedergabe, die Pause-Taste zu drücken. Die Wiedergabe wird pausiert. Ein Druck auf die Play-Taste bewirkt, dass die Wiedergabe fortgesetzt wird, jetzt jedoch zeitversetzt.

Mit den Tasten FF und REW kann innerhalb dem Zeitpunkt, an dem Pause gedrückt wurde und dem “Live” Zeitpunkt hin- und hergespult werden. Durch mehrmaliges Drücken der FF- und REW- Taste wird entsprechend schneller vor- oder zurückgespult. Während des Timeshiftingmodus ist die Kanalauswahl deaktiviert.

Wird die Stoptaste gedrückt, wird das Timeshifting verlassen und das Livebild abgespielt. Durch drücken der Play-Taste wird die Wiedergabe dort fortgesetzt, wo das Timeshifting verlassen wurde. Diese Verhalten ist vergleichbar mit einem Videorekorder, der beim Drücken der Stop-Taste das aktuelle TV-Programm anzeigt und durch Drücken der Play-Taste wieder dort zu spielen beginnt, wo er zuvor gestoppt wurde.

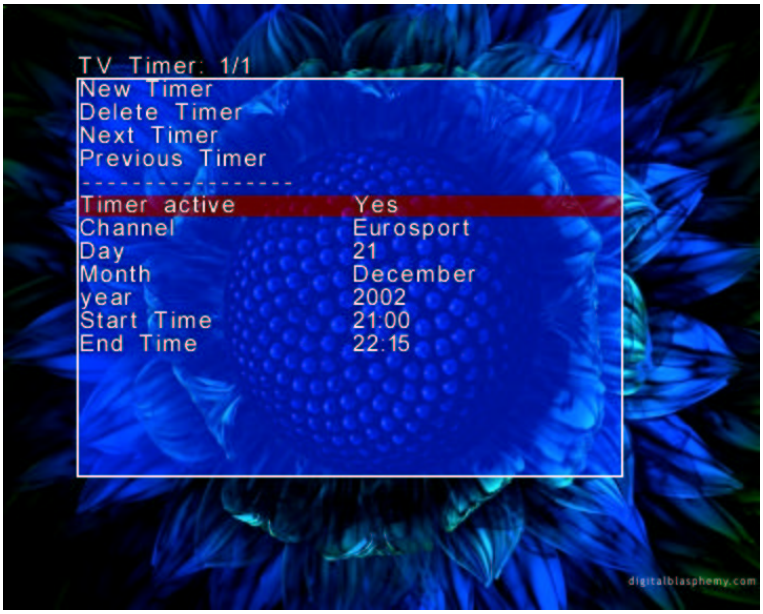

Abbildung 6.5: Der TV-Timer erlaubt die zeitgesteuerte Aufnahme von Fernsehprogrammen. Alle Daten wie Start-, Endzeit und Fernsehsender lassen sich per Menü eingeben.

6.1.6 TV-Timer

Fernsehprogramme können durch Verwendung des TV-Timers (Abbildung 6.5) zeitgesteuert aufgenommen werden. Wechselt man in das Timermenü, so hat man die Möglichkeit einen neuen Timer anzulegen, indem man auf den Eintrag “New Timer” wechselt und dort die Playtaste drückt. Daraufhin erscheinen neue Felder, in denen die gewünschte Aufnahme eingetragen werden muss:

Channel Hier wird der Sender ausgewählt, der aufgenommen werden soll. Mit den Navigationstasten links und rechts wird zwischen den Sendern gewechselt. Die Sendernamen werden aus dem VDR-Konfigurationsfile übernommen (vgl. Anhang A.4).

Day Gibt den Tag (1-31) an, an dem die Aufnahme gestartet werden soll. Der Tag kann mit Hilfe der Zahlentasten geändert werden.

Month Hier kann der Monat des Startzeitpunktes ausgewählt werden

Year Jahr, in dem die Aufnahme gestartet werden soll

Start Time Gibt die Uhrzeit an, um die die Aufnahme gestartet werden soll. Die Zeit wird mit den Zahlentasten eingegeben, dabei wird der Doppelpunkt, der die Stunden von den Minuten trennt automatisch eingefügt.

End Time Gibt die Uhrzeit an, um die die Aufnahme beendet werden soll.

Durch Drücken der Play-Taste innerhalb der oben aufgeführten Felder, wird der Timer übernommen. Zuvor wird überprüft, ob sich die eingestellten Zeiten mit einem anderen Timer überschneiden. In diesem Fall wird eine Fehlermeldung ausgegeben, die mit der Play-Taste bestätigt werden muss und der Timer wird nicht aktiviert.

Bei der Zeitangabe ist es möglich, nicht existierende Zeiten einzugeben, zum Beispiel der 31. September. Wird der Timer hinzugefügt, werden die Zeiten normalisiert, d.h. aus dem 31. September wird der 1. Oktober. Das gleiche gilt für die Zeitangabe. Die Uhrzeit, um die die Aufnahme gestoppt werden soll (End Time), darf kleiner als die State Time sein. In diesem Fall wird die Aufnahme erst am darauf folgenden Tag um die angegebene Uhrzeit gestoppt.

Alle aktiven Timer können angezeigt werden, indem man zwischen ihnen hin- und herblättert. Das wird durch Drücken der Einträge “Previous Timer” und “Next Timer” realisiert. Drückt man auf den Eintrag “Delete Timer”, so wird der gerade ausgewählte Timer gelöscht.

Da Anwendungen parallel ablaufen können, kann es vorkommen, das eine Aufnahme starten will, wenn man sich gerade im TV-Viewer befindet. Da beide Anwendungen die DVB-Karte benötigen, wird immer der TV-Viewer gestoppt, damit eine Aufzeichnung starten kann. Während einer Aufnahme kann somit auch nicht in den TV-Viewer gewechselt werden. Mit Hilfe der Playlist kann man sich aber die aktuell aufgenommenen Sendung anschauen.

6.1.7 Playlist

Die Playlist (Abbildung 6.6) bietet die Möglichkeit fast alle Arten von Multimediadateien abzuspielen oder sie in eine Liste einzutragen, wo sie nacheinander abgespielt werden. Es können dort aufgenommenen Fernsehprogramme abgespielt oder gelöscht werden. Momentan werden folgende Dateitypen unterstützt: AVI, MP3-Audio, CD-Audio und MPEG Audio/Video. Videos im AVI-Format können unterschiedlich kodiert sein, momentan wird der DivX-Codec unterstützt.

Die Playlist präsentiert sich in drei Fenstern. Zwischen diesen Fenstern kann mit Hilfe der Navigationstasten rechts und links gewechselt werden. Innerhalb der Fenster können mit den Navigationstasten hoch und runter Einträge ausgewählt werden.

Das Fenster unten links hat die Bezeichnung “Selection”. Hier wird das Quellmedium ausgewählt. Momentan sind das “Disk” und “CD”. Bei Auswahl von “Disk” (mit der Play-Taste) erscheint in dem rechten Fenster, das im folgenden “Tracklist” genannt wird, der Inhalt des MMBox-Root-Verzeichnisses (vgl. Anhang A.7.1). Bei der Auswahl “CD” erscheint dort die Trackliste einer eingelegten Audio-CD.

Im “Tracklist” Fenster kann der selektierte Eintrag durch Drücken der Play-Taste direkt abgespielt werden. Mit der Funktionstaste 1 kann das selektierte File oder CD-Titel in die Playliste eingefügt werden. Alle hinzugefügten Einträge

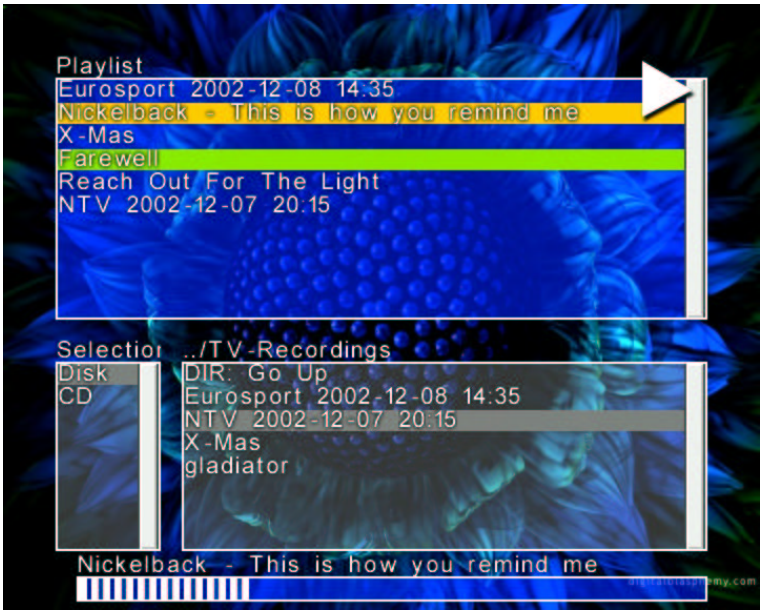

Abbildung 6.6: Die Playlist ist in 3 Fenster unterteilt. Das “Selection” Fenster bietet die Auswahl der Medienquellen wie Festplatte oder CD. Abhängig von der gewählten Medienquelle befindet sich im “Tracklist” Fenster dessen Inhalt. Diese Einträge können in das “Playlist” Fenster aufgenommen werden, dort werden sie der Reihe nach abgespielt. Der aktuell gespielte Titel wird farbig (orange) hervorgehoben.

werden sofort im Fenster “Playlist” sichtbar. Im “Tracklist” Fenster kann durch Drücken der Play-Taste in ein selektiertes Unterverzeichnis gewechselt werden. Hier können komplette Verzeichnisse und Dateien mit Funktionstaste 2 gelöscht werden, zuvor muss der Löschvorgang bestätigt werden.

Wird im “Playlist” Fenster die Play-Taste gedrückt, so wird der aktuell selektierte und alle nachfolgenden Einträge der Reihe nach abgespielt. Der aktuell gespielte Titel steht über der Fortschrittsanzeige und wird im “Playlist” Fenster hervorgehoben.

Funktionstaste 2 kann zum Löschen eines Eintrags benutzt werden. Funktionstaste 1 wechselt in einen “Editmodus”, der durch einen roten Selektionsbalken gekennzeichnet ist. Wird im Editmodus die Navigationstaste hoch oder runter benutzt, so wird der selektierte Eintrag nach oben oder unten verschoben. Durch nochmaliges Drücken der Funktionstaste 1 wird der Editmodus verlassen.

6.1.8 MP3-Spieler

Beim Starten des MP3-Spielers (Abbildung 6.7) erscheint eine Auswahlbox mit dem aktuellen Inhalt des MMBox-Root-Verzeichnisses (vgl. Anhang A.7.1). Mit

Abbildung 6.7: Trackauswahl des MP3-Spieles. Alle MP3-Dateien werden in einer Liste aufgeführt und können dort selektiert und abgespielt werden. Eine Fortschrittsanzeige zeigt den bisher gespielten Teil der Datei an.

den Navigationstasten kann eine MP3-Datei selektiert werden und mit der Play-Taste abgespielt werden. Nachdem die MP3-Datei abgespielt worden ist, wird die nächste MP3-Datei im aktuellen Verzeichnis abgespielt. Wird die Play-Taste auf einen Verzeichniseintrag gedrückt, so wird in dieses Verzeichnis gewechselt.

Der MP3-Player kann auch M3U-Dateien abspielen. M3U Dateien sind Playlistdateien für MP3-Files [85].

Eine Fortschrittsanzeige spiegelt die aktuelle Abspielposition innerhalb der gerade wiedergegebenen MP3-Datei wieder. Durch Drücken der FF- und REW-Tasten kann vor- oder zurückgespult werden.

Der MP3-Spieler ist auch Bestandteil des MMBox-Prototypen [65]. Der Playlist-Zustand bietet die gleiche Funktionalität wie der MP3-Spieler, jedoch wurde der MP3-Spieler wegen seiner leichten Bedienung auch in die MMBox-Anwendung integriert.

6.1.9 Taskmanager

Im Taskmanager werden alle laufenden Funktionen in einer Liste aufgeführt. In der Liste kann mit den Navigationstasten eine Anwendung selektiert werden. Beim Drücken der Play-Taste wird zur selektierte Anwendung gewechselt, in der man wie gewohnt navigieren kann.

Abbildung 6.8: Die Konfigurationsanwendung bietet Einstellungen, die das Verhalten der Anwendungen beeinflusst.

Funktionstaste 2 dient zum beenden von Anwendungen. Eine selektierte Anwendung in der Taskliste wird durch Drücken der Funktionstaste 2 sofort beendet.

6.1.10 Konfiguration

Im Konfigurations-Zustand (Abbildung 6.8) können Einstellungen vorgenommen werden, die das Verhalten der Anwendungen steuern. Beim Starten wird ein Auswahlfenster präsentiert, dessen Einträge mit den Navigationstasten hoch und runter selektiert werden können; mit den Tasten rechts und links wird der selektierte Eintrag verändert. Drückt man die Play-Taste, so werden alle geänderten Einstellungen übernommen und eine Bestätigungsmeldung ausgegeben. Folgende Einstellungen können vorgenommen werden:

AC3 Passthrough Ist dieser Eintrag auf “yes” gesetzt, wird beim Abspielen einer DVD AC3-Audiodaten direkt an die Soundkarte geleitet, wo sie dekodiert oder an einen externen Receiver weitergeleitet werden. In Abschnitt 5.2.3.1 wurde auf die AC3-Fähigkeit von Soundkarten näher eingegangen. Steht der Eintrag auf “no”, so wird ein Software-AC3-Decoder (Abschnitt 5.2.2.10) beim DVD-Spielen eingesetzt, der aus den AC3-Audiodaten, unkomprimierte Audiodaten generiert, die mit Soundkarten ohne AC3-Unterstützung abgespielt werden können.

Use CDDB Beeinflusst die Bezeichnungen der Tracknamen, die von CD-Player

und CD-Grabber beim Einlesen des CD-Inhaltes generiert werden. “yes” bewirkt, dass Informationen über Interpret und Titel aus einer Internetdatenbank bezogen werden (vgl. Abschnitt 6.1.4 und 6.1.3).

MP3 Enc. Bitrate Stellt die Bitrate ein, mit der der CD-Grabber die Audio-CD Titel nach MP3 konvertiert. Je höher die Bitrate ist, desto besser ist die Qualität der generierte MP3-Audiodatei. Eine hohe Bitrate bedeutet aber auch, dass mehr Daten in der MP3-Datei gespeichert werden müssen. Die generierten Dateien werden dadurch größer.

Use hardware MPEG decoder Einige DVB-Karten besitzen die Fähigkeit den empfangenen MPEG-Strom per Hardware zu dekodieren. Diese Option kann auf “yes” gestellt werden, damit der TV-Viewer im Live-Modus das Fernsehprogramm nicht per Software dekodieren muss (vgl. Abschnitt 4.2.8).

6.2 Anwendungs-Framework

Hauptidee für das Design der MMBox-Anwendung ist, dass sie leicht konfiguriert und modifiziert werden kann [65, 52]. Die Erweiterbarkeit steht ebenso im Vordergrund wie die leichte Anpassungsmöglichkeit des Erscheinungsbildes an persönliche Benutzervorlieben und die Anpassung der Struktur und des Verhaltens der Anwendung.

Es ist deshalb erforderlich, dass die Anwendung selbst angepasst und erweitert werden kann. Die Anwendung hat daher eine plugin-artige Struktur und kann mit Hilfe eines XML-Dokuments beliebig zusammengestellt werden.

Das XML-Dokument beschreibt die hierarchische Menüstruktur, wobei jedes Menü mehrere Einträge haben kann, die wiederum Untermenüs haben können oder die Funktionen (Zustände) wie CD-Spielen, MP3-Spielen aufrufen. Diese Zustände benutzen NMM-Knoten, die zu einem Flussgraph verbunden werden.

Über das XML-Dokument lassen sich die in Abschnitt 6.1 vorgestellten Zustände in die die Anwendung einbinden und konfigurieren.

Das Anwendungs-Framework lässt auch die parallele Ausführung mehrere Zustände zu, so dass beispielsweise eine Audio-CD nach MP3 konvertiert werden kann während eine Videodatei abgespielt wird.

6.2.1 Architektur der Hauptanwendung

Betrachtet man die einzelnen Zustände wie DVD-Spieler, TV-Viewer, usw., dann stellt man fest, dass alle Anwendungen unterschiedliche Ausgaben erzeugen und unterschiedliche Reaktionen beim Druck auf die gleichen Tasten zeigen. Drückt man im CD-Spieler auf die Pause-Taste, dann pausiert die CD-Wiedergabe. Beim Drücken der Pause-Taste im TV-Viewer startet jedoch der komplette Timeshifting-Mechanismus. Benutzereingaben müssen somit an die richtige Stelle geschickt werden, abhängig von dem aktuellen Zustand der Anwendung.

Zur Realisierung dieses Verhaltens wird das “Zustand” (engl. state) Design Pattern [22] benutzt. Dieses Design Pattern beschreibt wie ein Objekt sein verhalten ändern kann, wenn sich sein interner Zustand ändert. Die MMBox-Applikation wird als ein solches Objekt generiert, das sein Verhalten ändert, wenn verschiedene Funktionen ausgeführt werden. Diese Funktionen werden im folgenden als Zustände bezeichnet.

Die MMBox-Anwendung besteht aus mehreren Modulen:

- Anwendungs-Objekt (vgl. Abschnitt 6.2.2)
- Globale Knoten, inklusive OSDManagerNode (vgl. Abschnitt 6.2.3)
- XML Parser (vgl. Abschnitt 6.2.4)
- Zustände (vgl. Abschnitt 6.3)

6.2.2 Anwendungs-Objekt

Die Klasse `MMBoxApplication` repräsentiert die eigentliche MMBox-Anwendung mit allen globalen Knoten und Zuständen. Sie besitzt ein Interface für Benutzereingaben. Dabei gibt es jeweils eine oder mehrere Methoden für die in Abschnitt 6.1 aufgeführten Tasten:

```
class MMBoxApplication {
  /* Navigationstasten */
  Result Up();
  Result Down();
  Result Left();
  Result Right();

  /* Playtaste */
  Result Return();

  /* Backtaste */
  Result Back();

  /* Pausetaste */
  Result Pause();

  /* REW-Taste */
  Result Rewind();

  /* FF-Taste */
  Result Forward();

  /* Stoptaste */
  Result Stop();

  /* Funktionstaste 1 */
```

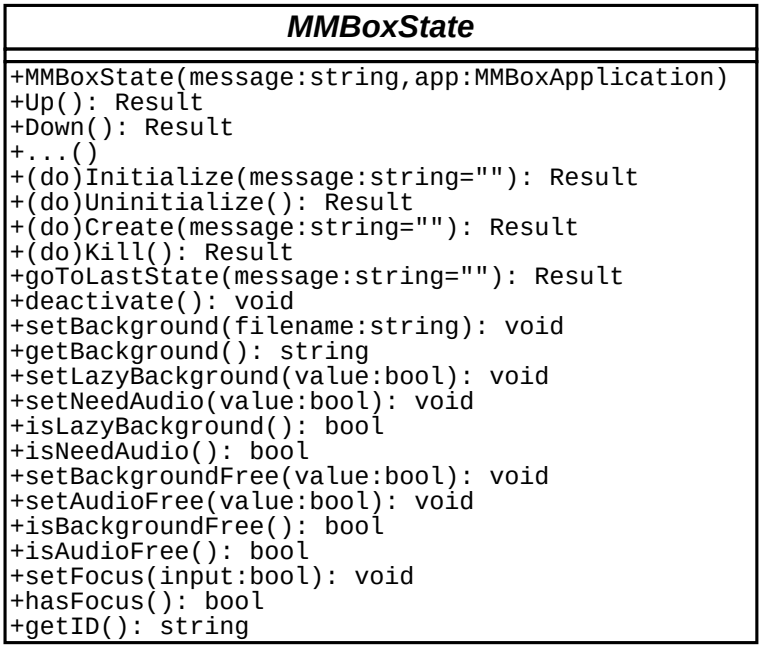

Abbildung 6.9: Public Methoden der MMBoxState Klasse.

```
  Result specialKey01();

  /* Funktionstaste 2 */
  Result specialKey02();

  /* Nummerntaste */
  Result numKey0();
  Result numKey1();
  Result numKey2();
  Result numKey3();
  Result numKey4();
  Result numKey5();
  Result numKey6();
  Result numKey7();
  Result numKey8();
  Result numKey9();
}
```

Im Anwendungs-Objekt werden zusätzliche Informationen gespeichert, wie die Namen und Aktivität der einzelnen Zustände. Das Anwendungs-Objekt regelt auch die Zustandsübergänge (vgl. Abschnitt 6.2.5). Jeder Zustand ist von der Klasse `MMBoxState` (Abbildung 6.9) abgeleitet. Momentan existieren 9 unterschiedliche Zustände (vgl. Abbildung 6.10).

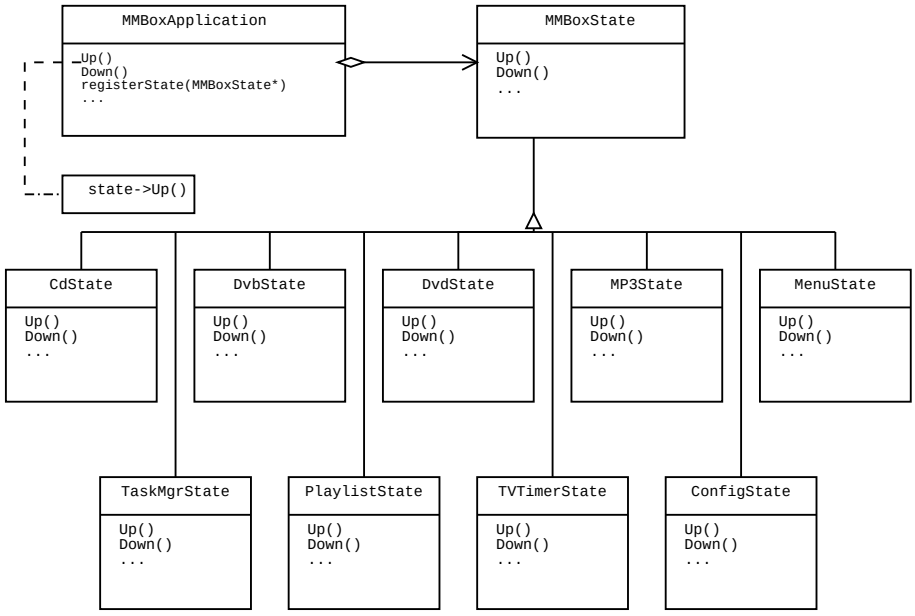

Abbildung 6.10: Das Klassendiagramm zeigt das Verhältnis zwischen verschiedenen Zuständen und der MMBox-Anwendung.

6.2.3 Globale Knoten

Die einzelnen Zustände benutzen NMM-Knoten und verknüpfen diese zu einem Flussgraphen, der bestimmte Multimediadaten verarbeiten kann. In jedem Zustand wird mindestens ein Senkeknoten verwendet (vgl. Abschnitt 5.2.3). Einige Zustände geben Audiodaten wieder und benötigen einen PlaybackNode, andere geben Videodaten wieder und brauchen einen XDisplayNode oder gegebenenfalls beide Senkeknoten. Zustände, die Auswahlboxen, Fortschrittsanzeigen und andere Benutzerinformationen ausgeben brauchen einen OSDManagerNode.

Aus diesem Grund sind diese drei Knoten Bestandteil des MMBox-Anwendungs-Objekts. Sie sind permanent vorhanden und müssen von einem Zustand nicht erstellt werden, sondern können von dem Anwendungs-Objekt angefordert werden. Ein Zustand kann sich direkt mit einem globalen Knoten verbinden, sofern der globale Knoten nicht schon mit einem anderen Zustand verbunden ist (vgl. Abbildung fig:standardgraph).

Damit Zustände Informationen auf ein Videobild ausgeben können, müssen sie aber nicht unbedingt mit dem OSDManagerNode verbunden zu sein. Sie können Windows und Widgets generieren, die zum OSDManagerNode hinzugefügt werden und direkt angezeigt werden. Zustände, die keinen Hintergrund benötigen, können sich mit der Methode

```
createDefaultGraph( background )
```

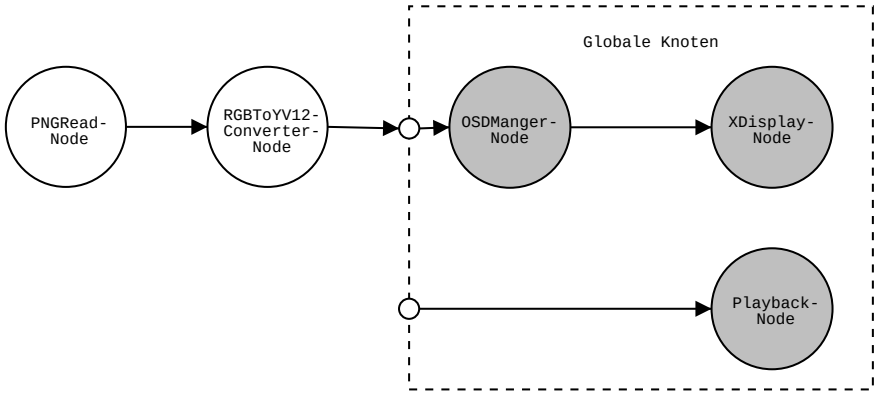

Abbildung 6.11: Aufbau eines “Standardgraphen”, der von den Zuständen angefordert werden kann.

der `MMBoxApplication` Klasse einen Graphen wie in Abbildung 6.11 dargestellt anfordern, der mit Hilfe des PNGReadNode (Abschnitt 5.2.1.4) ein Bild an den XDisplayNode schicken kann. Dieser Graph wird im folgenden *Standardgraph* genannt.

6.2.4 XML Parser

Zustände können unterschiedliche Schnittstellen zur Konfiguration haben. Ein Zustand, der für eine Menüanzeige verantwortlich ist, braucht Informationen, wie Knöpfe angeordnet werden sollen, wie das Hintergrundbild aussehen muss etc. Ein Zustand, der dem Benutzer eine Auswahlbox präsentiert, braucht Informationen über Größe, Position und Farbe der Auswahlbox.

Der XML Parser liest ein XML Dokument ein, in dem alle Zustände beschrieben sind, d.h. u.a. ihre Menüstruktur oder ihr Verhalten. Der XML Parser ist der zentrale Teil, in dem die eigentlichen Instanzen der Zustände erzeugt werden. Er extrahiert zustands-spezifische Informationen und gibt sie an den Zustand weiter. Wird ein neuer Zustand erstellt, so muss im XML Parser der Programmcode platzieren werden, der Informationen aus dem XML Dokument an den Zustand liefert (Abbildung 6.12). Der XML Parser verwendet die xmlpp-Library [4] zum parsen von XML Dokumenten.

Ein XML Dokument für die MMBox-Anwendung, das u.a. die Zustände beschreibt, hat folgenden Aufbau:

```
<?xml version="1.0"?>
<configuration>
  <resourcepath>/usr/local/share/nmm/mmbox/</resourcepath>

  <state id="MainMenu" background="image.png"
       needvideo="no" needaudio="no">
```

```
  </state>
</configuration>
```

Alle Informationen werden durch ein “configuration” Tag geklammert. Der “resourcepath” Eintrag gibt einen Pfad an, der als Präfix für die im XML Dokument benutzten Dateien benutzt wird. Zustände werden durch einen Zustands-Tag geklammert, für einen MP3State könnte er zum Beispiel “mp3play” lauten. Nach dem Zustands-Tag folgen Standardattribute, die für jeden Zustand benutzt werden:

id Gibt einen eindeutigen String an, mit dem der Zustand identifiziert werden kann.

background bezeichnet eine PNG-Datei, die als Hintergrund benutzt werden soll. Die Datei wird relativ zum Resourcepath gesucht.

needvideo Steht auf “no”, wenn der Zustand keinen Hintergrund, also keinen XDisplayNode benötigt.

needaudio Steht auf “no”, wenn der Zustand keine Audioressource, also keinen PlaybackNode benötigt.

Der Zustands-Tag kann durch eigene Attribute erweitert werden, ebenso können eigene Tags zwischen dem Zustands-Tag definiert werden. Wichtig ist, dass der XML Parser so erweitert wird, dass er die Attribute und Tags verarbeiten kann.

6.2.5 Integration der Zustände in die Anwendung

Einzelne Instanzen der Zustände werden durch den XML Parser generiert und konfiguriert (vgl. Abschnitt 6.2.4). Die MMBox-Anwendung kennt alle vom XML Parser generierten Zustände. Abbildung 6.13 zeigt wie ein Zustand in die Anwendung integriert wird.

Benutzereingaben werden von der Applikation an den momentan aktivierten Zustand weitergeleitet. Der Zustand selbst hat die Möglichkeit sich mit den globalen Knoten zu verbinden. In Abbildung 6.13 werden alle Benutzereingaben an den DvdState weitergeleitet, der mit den globalen Knoten verbunden ist, da er sowohl Audiodaten als auch Videodaten verschickt (vgl. Abschnitt 6.3.2). Da die

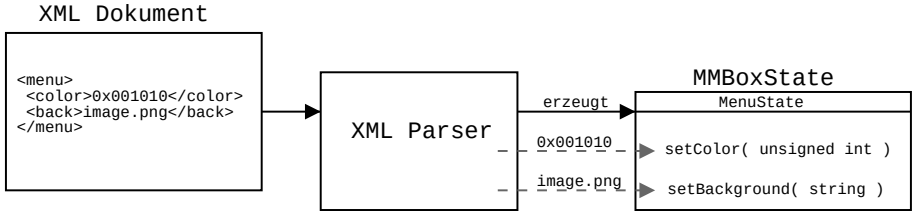

Abbildung 6.12: Der XML Parser erzeugt Zustände und liest zustand-spezifische Informationen aus einem XML File und leitet sie an den Zustand weiter.

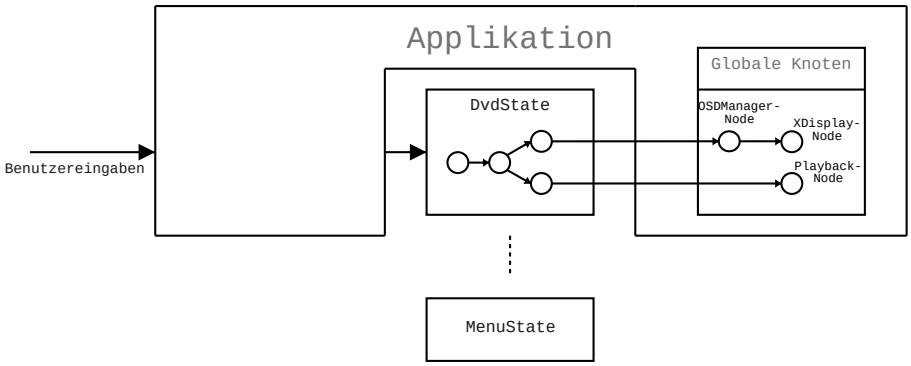

Abbildung 6.13: Architektur der MMBox-Applikation

Benutzereingaben direkt an den Zustand geleitet werden, kann das Verhalten des Zustandes durch Tastendrücke beeinflusst werden. Es können beispielsweise Aktionen wie Pause oder Kapitelwechsel durchgeführt werden.

Der Zustand kann jederzeit durch einen anderen Zustand ersetzt werden. Wird der DvdState beendet, so wird er von den globalen Knoten getrennt und z.B. durch einen MenuState ersetzt (vgl. Abschnitt 6.3.1). Der MenuState kann sich jetzt mit den globalen Knoten verbinden und erhält die Benutzereingabe. Benutzereingaben bewirken in diesem Zustand andere Reaktionen als im DvdState. Hier kann zwischen verschiedenen Menü und Untermenüs navigiert werden oder andere Zustände gestartet werden.

Da die Kommunikation mit dem Zustand indirekt über die MMBox-Anwendung läuft, scheint die Anwendung je nach Zustand ihr Verhalten zu ändern (State-Design Pattern [22]).

Paralleles Ausführen mehrerer Zustände

Die globalen Knoten können nur von jeweils einem Zustand belegt werden. Es gibt jedoch Zustände, die nicht alle oder keine globalen Knoten brauchen, beispielsweise ein GrabState, der eine Audio-CD nach MP3 konvertiert und auf Festplatte speichert (vgl. Abschnitt 6.3.4). In diesem Fall kann der GrabState weiterlaufen während z.B. eine Videodatei mit dem PlaylistState wiedergegeben wird (vgl. Abschnitt 6.3.7).

Abbildung 6.14 zeigt eine laufende DVD Anwendung, die jedoch nicht den Eingabefocus besitzt. Der Eingabefocus ist einem MenuState zugeordnet, der das Haupt- oder Untermenü einblendet. Während der DVD Wiedergabe kann in dem Menü navigiert werden und in eine andere Anwendung gewechselt werden. Der MenuState braucht zur Darstellung der Menüpunkte nur einen OSDManagerNode, dem er seine Windows zum Überblenden auf das aktuelle Videobild übergibt.

Welche globalen Knoten ein Zustand benötigt, wird im XML Dokument angegeben, das die Zustände beschreibt (vgl. Abschnitt 6.2.6).

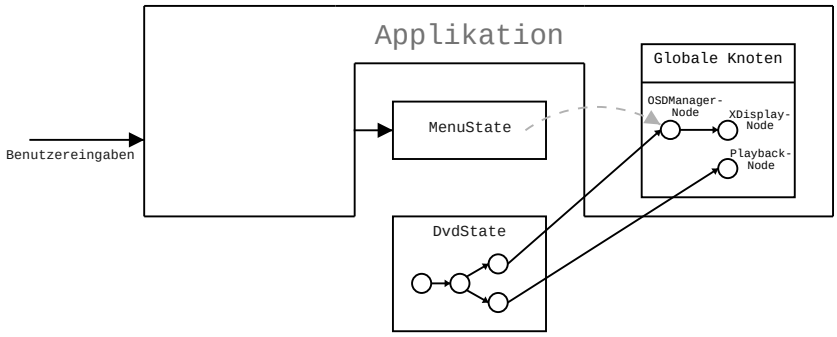

Abbildung 6.14: MMBox-Applikation mit mehreren aktiven Zuständen, dabei hat jeweils ein Zustand den Eingabefocus.

Weiterleiten der Benutzereingaben

In Abschnitt 6.2.2 wurden die Navigations-Methoden der `MMBoxApplication` Klasse vorgestellt. Wird eine solche Methode aufgerufen, so wird die entsprechende Methode des Zustandes aufgerufen, der gerade den Eingabefocus besitzt.

Die Methoden der `MMBoxApplication` Klasse werden direkt über einen Dispatcher (Abschnitt 3.3) aufgerufen, der seine Events von einem Producer (Abschnitt 5.3) erhält. Beim Dispatcher müssen alle Events registriert werden, die die Producer für die Tastendrücke erzeugen:

```
(EventDispatcher*) dispatcher->registerEvent("KEY_Up",
  new TEDObject0<MMBoxApplication>(
                    (MMBoxApplication*) app,
                    &MMBoxApplication::Up));
```

Der “KEY_Up” String wird sowohl vom XProducer (Tastatur), als auch vom LircProducer (Fernbedienung) erzeugt, die noch mit dem Dispatcher verbunden werden müssen:

```
Xproducer.registerXDisplayNode(
  (XDisplayNode*) app->getDisplayNode());
Xproducer.readConfig("configurationfile.xml");
Xproducer.connectTo((EventDispatcher*) dispatcher);
Xproducer.start();
LircProducer.readConfig("remote.conf");
LircProducer.connectTo(dispatcher);
LircProducer.start();
```

Die Registrierung der Events und Verknüpfung der Producer mit einem Dispatcher müssen manuell ausgeführt werden. Abbildung 6.15 zeigt wie ein Tastendruck letztendlich die Ausführung einer Statemethode bewirkt.

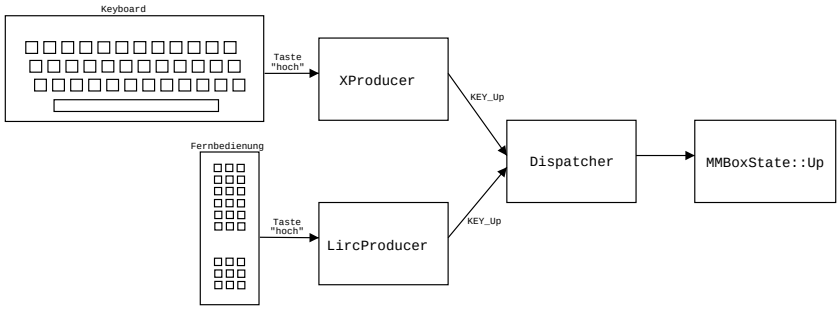

Abbildung 6.15: Benutzereingaben werden durch ihre Producer abgefangen, in Events umgewandelt und zum Dispatcher geleitet, der eine registrierte Methode aufruft

6.2.6 Statewechsel

Damit ein Zustand eindeutig identifiziert werden kann, wird ihm beim Erstellen (im Konstruktor) eine ID zugeordnet. Erstellt wird ein Zustand von einem XML Parser (Abschnitt 6.2.4). Zustände können zwei Zustände haben: aktiv und inaktiv. Ein aktiver Zustand ist gestartet und führt seine Funktionen aus (z.B. MP3-Abspielen, CD-Grabben). Ein inaktiver Zustand wurde beendet oder noch nie gestartet. Ein aktiver Zustand hat nicht direkt den Eingabefocus. Um zu testen, ob ein Zustand den Eingabefocus besitzt, kann er die Methode `hasFocus()` aufrufen, die entweder **`true`** oder **`false`** zurückliefert. Bei einem Zustandswechsel, d.h. beim Wechsel des Eingabefocus werden abhängig vom jeweiligen Zustand, verschiedene Methoden aufgerufen. Zuerst wird die `uninitialize()` Methode des Zustandes aufgerufen, der verlassen wird, ist der Zustand bereits inaktive, wird stattdessen seine `kill()` Methode aufgerufen. Die Initialisierung des Zustandes, zu dem gewechselt wird, ist auch abhängig von dessen Zustand. Bei aktiven Zuständen wird dessen `initialize( string )` Methode aufgerufen, bei inaktiven Zuständen die `create( string )` Methode. Diesen Methoden kann noch ein String übergeben werden, der eine Nachrichten für den Zustand enthalten kann. Die Unterscheidung der Methodenaufrufe für aktive und inaktive Zustände erleichtert deren Implementierung. Inaktive Zustände können bei ihrer Aktivierung Ressourcen anfordern, Knoten erstellen usw. Beim Wiederbetreten eines schon aktiven Zustände müssen diese Schritte meist nicht mehr durchgeführt werden, hier werden meist nur zustand-spezifische Informationen aktualisiert und mit dem OSDManagerNode angezeigt. Ein Stateprogrammierer implementiert jedoch diese Methoden nicht direkt, sondern ihre Template-Methoden (vgl. Abschnitt 3.5):

```
Result doInitialize( string );
Result doUninitialize();
Result doCreate( string );
Result doKill();
```

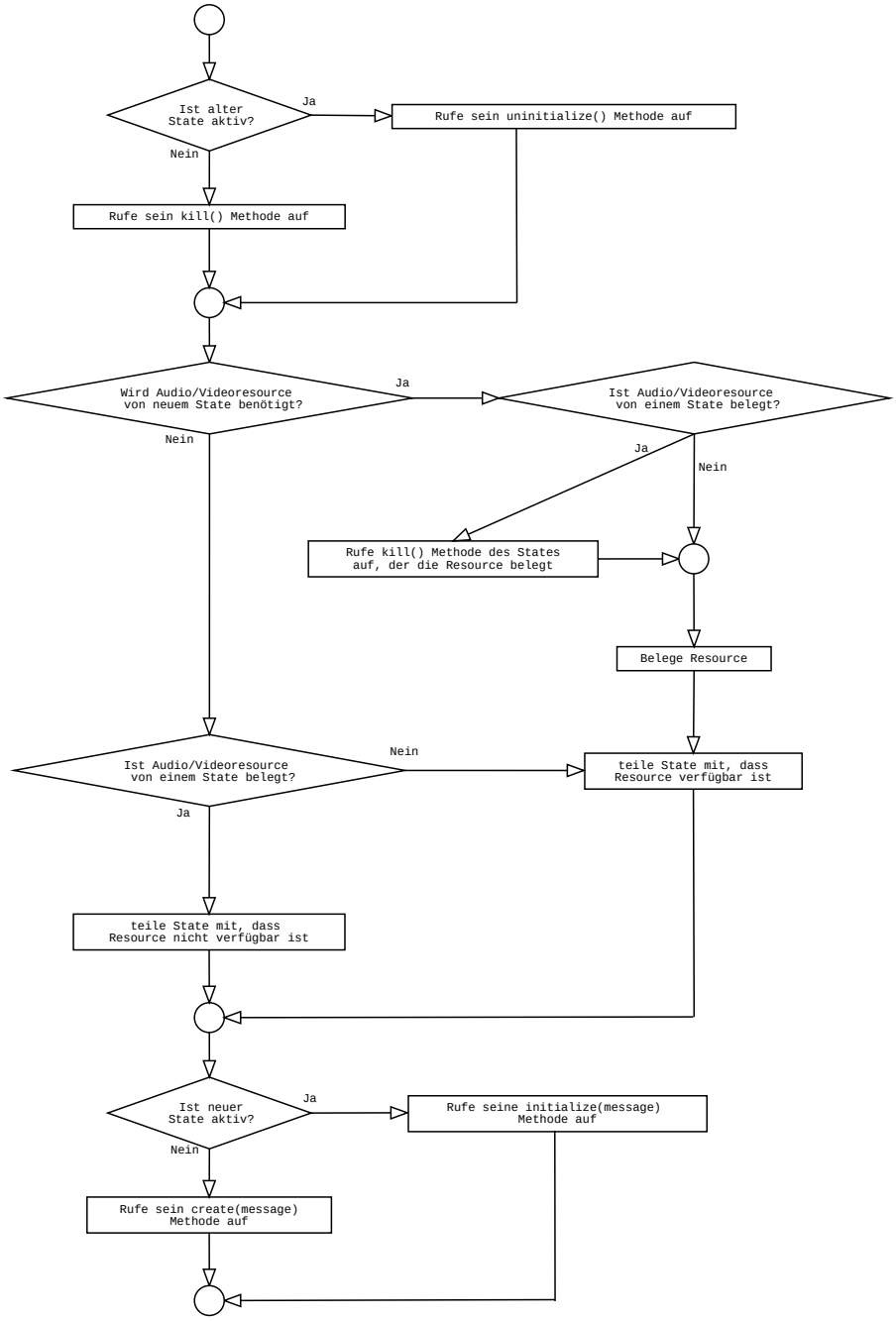

Abbildung 6.16: Flussdiagramm eines “changeState” Aufrufs. Der neue Zustand bekommt Informationen über Verfügbarkeit von Audio- und Videoressource mitgeliefert, eventuell wird ein Zustand, der die Ressource belegt, beendet

Abbildung 6.17: Durch Änderung des XML Dokuments, das das Hauptmenü der MMBox beschreibt, können verschiedenen Oberflächen (Skins) erzeugt werden. Die hier gezeigten Skins haben die Namen “Neon3” und “Neon2”.

Die globalen Knoten OSDManagerNode und PlaybackNode sind Ressourcen die jeweils nur mit einem einzigen Zustand verbunden werden können. Deshalb wird in der `MMBoxApplication` vermerkt, welche Zustände diese Ressourcen gerade benötigen. Zustände, die auf die Darstellung von Video oder Bilder verzichten können, setzen das “needvideo” Flag auf “no”. Kann ein Zustand auf den PlaybackNode verzichten, so wird ein “needaudio” Flag auf “no” gesetzt. Diese Flags müssen im XML Dokument, in dem die Zustände konfiguriert werden, angegeben werden.

Der Statewechsel wird von der `MMBoxApplication` Klasse koordiniert. Über die Methoden `setBackgroundFree(bool)` und `setAudioFree(bool)` kann den Zuständen mitgeteilt werden, ob die Audio- oder Videoressource gerade benutzt wird. Der XML Parser ruft diese Methoden mit dem im XML Dokument angegeben werden auf. Zum Wechseln eines Zustandes wird die Methode

```
changeState( string stateid, string message )
```

aufgerufen, die die ID des neuen Zustandes erwartet und eine Nachricht, die diesem Zustand beim Initialisieren übergeben wird. Abbildung 6.16 zeigt den Ablauf eines `changeState` Aufrufs.

6.3 Implementierung der Zustände

In den nächsten Abschnitten wird der Aufbau aller in Abschnitt 6.1 genannten Zustände beschrieben und deren Konfiguration durch eine XML-Datei. Die komplette XML-Datei befindet sich im Anhang A.7.2. Die Bezeichnung der Zustände richten sich dabei nach ihren Klassennamen.

6.3.1 AutoMenu

Der Zustand “AutoMenu” generiert und verwaltet die in Abschnitt 6.1.1 vorgestellten Menüs und Untermenüs. Menüs werden durch Radiobuttons realisiert, d.h. es

steht eine Auswahl von mehreren Knöpfen zur Verfügung, von denen aber nur einer aktiviert werden kann. Knöpfe werden durch Grafiken dargestellt, die je nach Zustand (selektiert, unselektiert) eine andere Gestalt annehmen. Für einen aktivierten Knopf kann keine Grafik angegeben werden, da die Aktivierung unmittelbar eine Aktion auslöst, so dass der aktivierte Knopf nie sichtbar wäre (vgl. Abbildung 6.17). Zur Speicherung der Grafiken wird ein Bitmapreader (Abschnitt 5.1.5) benutzt. Die Anordnung und Ansteuerung der Knöpfe wird durch ein Radiobutton-Widget (Abschnitt 5.1.6) realisiert. Dieses Widget bietet schon die komplette Funktionalität, um ein Menü darzustellen und um einzelne Buttons zu selektieren.

Im Konstruktor des Zustandes kann ein Bitmapreader angegeben werden:

```
AutoMenu( string id, MMBoxApplication* app, BitmapReader*
    bmreader );
```

Die Schnittstelle zum Hinzufügen von Knöpfen ist identisch mit dem Interface des Radiobutton-Widgets. Aufrufe von `addButton`, `setStep`, `setDimension` werden lediglich an das Widget weitergeleitet (vgl. Fassade-Design-Pattern [22]). Das Radiobutton-Widget erwartet u.a. ein Label bei der `addButton`-Methode, dort wird die ID des Zustandes eingetragen, der beim Drücken des Knopfes gestartet werden soll. Beim Starten der Applikation wird zuerst der Zustand mit der ID “MainMenu” gestartet. Das XML-Dokument muss ein “menu” Tag erhalten, damit ein Menü generiert wird:

```
<menu id="MainMenu" columns="1" background="back.png"
     needvideo="no" needaudio="no">

  <entry index = "1"
    on = "dvd.on.png"
    off = "dvd.off.png"
    x  = "30"
    y  = "150">DVDMenu</entry>

  <entry index = "2"
    on = "playlist.on.png"
    off = "playlist.off.png"
    x  = "180"
    y  = "40">Playlist</entry>

</menu>
```

Neben den Standardattributen erwartet das “menu” Tag ein “column” Attribut, das die Anzahl Spalten angibt, aus dem das Menü besteht. Mit dieser Information kann mittels der Navigationstasten hoch und runter direkt in eine andere Zeile des Menüs gesprungen werden (vgl. Abbildung 6.18). Das “needvideo”und “needaudio” Tag sollte bei Menüs immer auf “no” stehen, da Menüs keine Audioressource oder Hintergrund benötigen. Ist der Hintergrund nicht belegt, so wird das Bild, das durch das “background” Tag angegeben wurde mit Hilfe des Standardgraphen angezeigt.

Innerhalb der “menu” Tags werden die einzelnen Menüknöpfe beschrieben.

Abbildung 6.18: Ein Menü kann aus einer oder mehreren Spalten bestehen. Die hier gezeigten Skins haben die Namen “Timer” und “Icon”.

Jeder Knopf hat einen “entry” Tag. Der Inhalt diese Tags bezeichnet die ID eines Zustandes, der gestartet werden soll, wenn der Knopf gedrückt wurde. Das kann wiederum ein Menü sein oder eine konkrete Anwendung (CD-Player, MP3-Player usw.). Die Attribute beschreiben die Gestalt und Anordnung des Knopfes:

index Dient als Navigationshilfe. Beim Drücken der Navigationstasten links oder rechts wird der Index um eins erniedrigt oder erhöht und der Knopf mit dem resultierenden Index selektiert. Beim Drücken der Navigationstasten runter oder hoch wird der Index um den Wert “column” erniedrigt oder erhöht. Verwendung findet dieses Attribut bei Menüs, die mehrere Spalten haben. Der Wert des Attributs wird auf die Anzahl der Spalten gesetzt. Damit wird erzielt, dass mit den Navigationstasten hoch und runter korrekt zwischen den Zeilen navigiert werden kann.

off Bezeichnet eine PNG-Grafik, die den Knopf in seinem unselektierten Zustand zeigt.

on Bezeichnet eine PNG-Grafik, die über einen Knopf gezeichnet wird, wenn dieser selektiert wurde.

x X-Koordinate der oberen linken Ecke der PNG-Grafik.

y Y-Koordinate der oberen linken Ecke der PNG-Grafik.

Der Zustand “AutoMenu” deaktiviert sich beim Statewechsel automatisch, damit wird verhindert, dass sich Menü überlagern.

6.3.2 DvdState

Der DvdState implementiert den in Abschnitt 6.1.2 vorgestellten DVD-Player. Zur Realisation dieses Players werden neben den globalen Knoten noch 6 weitere Knoten benötigt:

- DVDNavReadNode (Abschnitt 5.2.1.5)

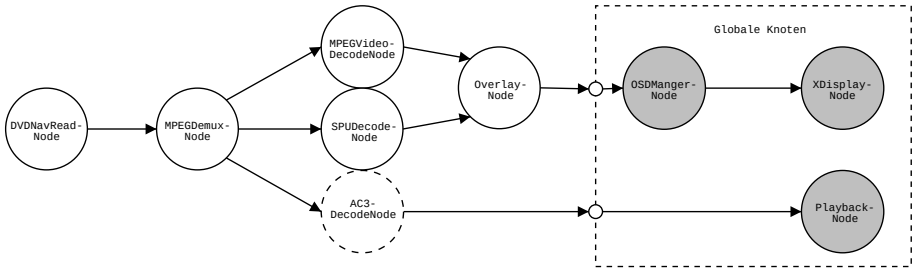

Abbildung 6.19: Aufbau eines NMM-Graphen für einen DVD Player

- MPEGDemuxNode (Abschnitt 5.2.2.4)
- MPEGVideoDecodeNode (Abschnitt 5.2.2.6)
- SPUDecodeMode (Abschnitt 5.2.2.2)
- OverlayNode (Abschnitt 5.2.2.3)
- AC3AudioDecodeNode (Abschnitt 5.2.2.10)

Abbildung 6.19 zeigt, wie die Knoten miteinander verbunden sind. Der DVDNavReadNode liest Daten von einer Video-DVD und schickt diese an den MPEGDemuxNode, wo Audio-, Video und Untertitelströme voneinander getrennt werden. Der MPEG-komprimierte Videostrom wird vom MPEGVideoDecodeNode in einzelne Bilder dekodiert. Der SPUDecodeNode dekodiert die Hervorhebungsbereiche der Menüknöpfe und die Untertitel. Diese SPU-Bilder werden im OverlayNode auf das aktuelle Videobild geblendet und danach zu den globalen Knoten geschickt, wo sie letztendlich angezeigt werden. Abhängig von der Konfiguration werden die AC3 komprimierten Audiodaten zuerst an den AC3AudioDecodeNode geschickt oder direkt an den PlaybackNode, wenn dieser die direkt AC3-Ausgabe unterstützt. Die Kommunikation läuft über Events. Der DVDNavReadNode reagiert auf Navigationstasten und schickt seinerseits wieder Kontrollinformation zum MPEG-DemuxNode, um dort Audio- und SPU Strom auszuwählen und zum SPUDecodeNode und OverlayNode, um Untertitel und Menüknöpfe anzuzeigen. Damit ein DvdState erzeugt wird, muss das XML-Dokument angepasst werden:

```
<dvdplay id="DVDPlayer"
         needvideo="yes" needaudio="yes">
</dvdplay>
```

Das Attribut “needvideo” ist hier auf “yes” gesetzt ebenso das “needaudio” Attribut, da der DVD Player sowohl die Audioressource benötigt, als auch Videodaten auf dem Hintergrund ausgibt.

6.3.3 CdState

Der in Abschnitt 6.1.3 vorgestellte CD Player wird durch den CdState implementiert. Er besteht neben dem globalen PlaybackNode aus 2 weiteren Knoten:

- CDDANode (Abschnitt 5.2.1.1)
- CopyNode (schickt eingehende Buffer an mehrere Ausgänge)

Der Zustand instantiiert einen CDDANode (Abschnitt 5.2.1.1), verbindet ihn mit einem CopyNode, der die eingehenden Daten dupliziert und an seine zwei Ausgänge leitet (Abbildung 6.20). Ein Ausgang ist mit dem PlaybackNode verbunden. Der zweite Ausgang des CopyNode ist mit dem zweiten Eingang des OSDManagerNode verbunden. Der OSDManagerNode extrahiert aus den eingehenden Buffern nur ihre Zeitstempel und stellt sie mit Hilfe eines TimeView-Widgets dar. Der CDDANode erstellt die Zeitstempel und beginnt bei Zeit "null", wenn ein neuer CD-Titel gespielt wird. Somit wird die genaue Spielzeit des aktuellen CD-Titels angezeigt. Da ein CD Player keine Videodaten produziert, wird ein Standardgraph erstellt, der das Hintergrundbild anzeigt. Wenn der Zustand verlassen wird, der aktuelle CD-Titel aber noch weitergespielt werden soll, wird der Ausgang des CopyNode, der mit dem OSDManagerNode verbunden ist, gekappt. Der CopyNode ist so implementiert, dass er Buffer, die an einen unverbundenen Ausgang geschickt werden sollen, löscht.

Es werden zwei Einträge aus dem MMBox Konfigurationsfile (Anhang A.7.1) ausgewertet. Zum einen der "cddb" Eintrag, der auf "true" gesetzt werden muss, damit die Titelinformationen über eine Internetdatenbank abgefragt werden. Der zweite Eintrag lautet "cddbserver", der die Adresse des Datenbankservers angibt.

Zum Integrieren des CdState in die Applikation, muss auch hier wieder das XML Dokument angepasst werden:

```
<cdplay id="CDPlayer" background="back.png"
        needvideo="no" needaudio="yes">
</cdplay>
```

Teile dieses Zustandes wurden aus [65] entnommen.

6.3.4 GrabState

Der GrabState realisiert den in Abschnitt 6.1.4 gezeigten CD-Grabber. Er besteht aus 3 Knoten:

- CDDANode (Abschnitt 5.2.1.1)
- MPEGAudioEncodeNode (Abschnitt 5.2.2.8)
- GenericWriteNode (Schreibt empfangende Buffer auf die Festplatte)

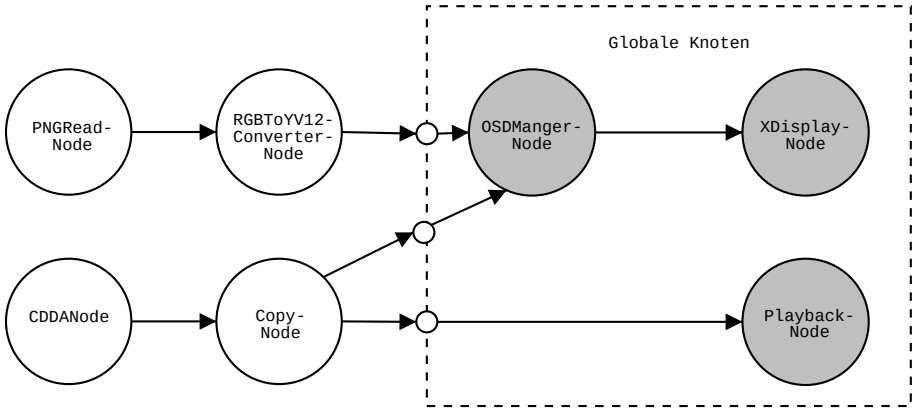

Abbildung 6.20: Aufbau eines NMM-Graphen für einen CD Player

Er verknüpft die Knoten zu einem Flussgraphen (Abbildung 6.21). Der CDDA-Node liest Audiodaten von einer CD und leitet sie an den MPEGAudioEncode-Node weiter, wo die Daten nach MP3 konvertiert werden. Die konvertierten Daten werden mit Hilfe des GenericWriteNode auf Festplatte geschrieben. Der GrabState reagiert wie der CdState auf die Konfigurationseinträge “cddb” und “cddbserver”, um gegebenenfalls die CD-Titelinformationen aus einer Internetdatenbank zu nehmen.

Die MP3-Files werden relativ zum Rootpath in ein Verzeichnis geschrieben, das durch den Eintrag “grabdestinationdirectory” der Konfigurationsdatei spezifiziert wird. Dort wird für jede CD ein eigenes Verzeichnis angelegt. Zusätzlich kann die Aufnahmequalität durch folgende zwei Einträge bestimmt werden:

- “grabmp3bitrate” Bestimmt die Bitrate, mit der das MP3 enkodiert werden soll. Mit 128 KBits wird nahezu CD-Qualität erreicht. Bei einem Wert von 192 KBits ist die Qualität von der einer CD nicht mehr zu unterscheiden.
- “grabmp3quality” Bestimmt die Qualität der Aufnahme, sie wird durch eine Nummer von 0 bis 9 angegeben, wobei eine hohe Zahl für bessere Qualität steht.

Der Eintrag in der XML-Datei lautet:

```
<cdgrab id="CDGrabber" background="back.png"
        needvideo="no" needaudio="no">
</cdgrab>
```

Teile des GrabState wurden aus [65] entnommen.

6.3.5 DvbState

Der DvbState implementiert den in Abschnitt 6.1.5 vorgestellten TV-Viewer. Hierzu wird ein Graph mit bis zu 7 Knoten aufgebaut:

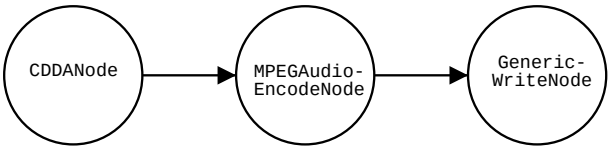

Abbildung 6.21: Aufbau eines NMM-Graphen für einen CD Grabber

- DVBReadNode im YV12-Modus (Abschnitt 5.2.1.2)
- DVBReadNode im MPEG-Modus (Abschnitt 5.2.1.2)
- MPEGTimeshiftingNode (Abschnitt 5.2.2.5)
- MPEGDemuxNode (Abschnitt 5.2.2.4)
- MPEGVideoDecodeNode (Abschnitt 5.2.2.6)
- MPEGAudioDecodeNode (Abschnitt 5.2.2.9)
- DevNullNode (der einkommende Buffer einfach löscht)

Abbildung 6.22 zeigt den NMM-Graphen für den TV-Viewer, so wie er bei allen DVB-Karten aufgebaut werden kann. Der DVBReadNode liefert hier immer einen MPEG-Strom, der im TimeshiftingNode verwaltet wird. Dieser Knoten schreibt bei aktiviertem Timeshifting Modus die MPEG Daten auf Festplatte und bietet Mechanismen zum Vor- und Zurückspulen. Der Konfigurationseintrag “hard_mpeg=no” zwingt den Zustand dazu, immer einen solchen Graphen aufzubauen.

Bei DVB Karten mit eingebautem Hardwaredekoder kann dieser Eintrag auf “yes” gesetzt werden. Im Live-Modus wird dann ein Graph wie in Abbildung 6.23 aufgebaut. Die Videobilder des DVBReadNode können direkt an die globale Senke

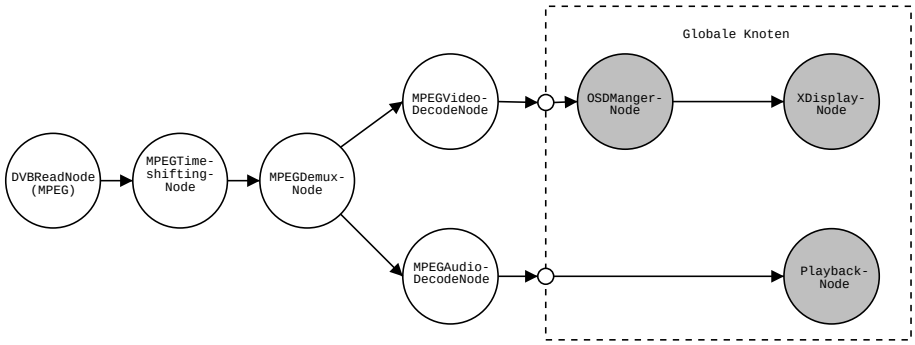

Abbildung 6.22: Aufbau eines NMM-Graphen für den TV Viewer im Timeshifting-Modus und Live-Modus ohne Hardwarebeschleunigung

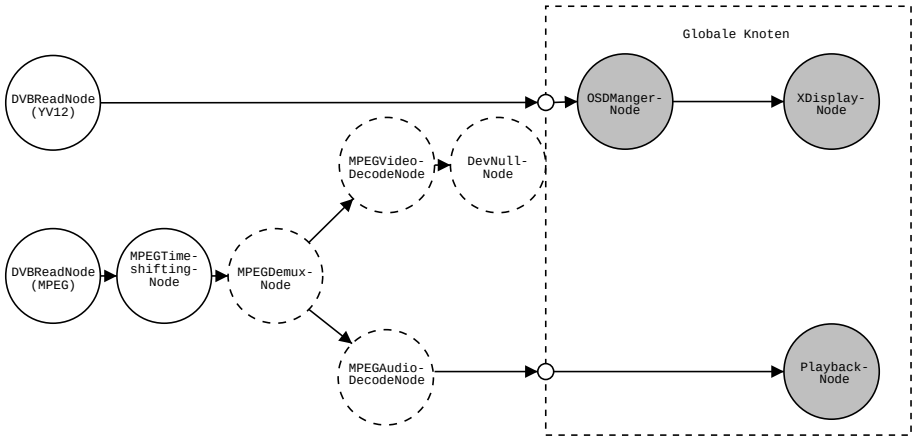

Abbildung 6.23: Aufbau eines NMM-Graphen für den TV Viewer im Live-Modus mit Hardwarebeschleunigung.

geschickt werden. Das Audiosignal wird durch ein Loopback-Kabel zur Soundkarte geleitet. Eine zweite Instanz des DVBReadNode liefert aber nach wie vor einen MPEG-Strom an den TimeshiftingNode, da nur der MPEG-Strom platzsparend auf Platte gesichert werden kann. Der TimeshiftingNode liefert hier aber keinen Daten an seinen Nachfolger (MPEGDemuxNode). Startet das Timeshifting (durch Drücken der Pausetaste), so wird der Graph verbunden wie er in Abbildung 6.23 zu sehen ist.

Der Eintrag für den DvbState in der XML-Datei lautet:

```
<dvbplay id="DVB-TV"
        needvideo="yes" needaudio="yes">
</dvbplay>
```

Teile dieses Zustandes wurden aus [65] entnommen.

6.3.6 MP3State

Der in Abschnitt 6.1.8 gezeigte MP3-Player wird durch den MP3State implementiert. Da der MP3-Player nur die Audio-Ressource benötigt, aber keinen Hintergrund, kann er sich den in Abschnitt 6.2.3 vorgestellten Standardgraphen anfordern, der bei Bedarf ein Hintergrundbild anzeigt (Abbildung 6.24). Der MP3ReadNode (Abschnitt 5.2.1.3) liest eine MP3-Datei von Festplatte und schickt die Daten an den MPEGAudioDecodeNode (Abschnitt 5.2.2.9), der die Daten dekodiert und an den PlaybackNode schickt.

Der MP3State registriert sich selbst als Listener (Abschnitt 3.3) für ein *progress* Event, das vom MP3ReadNode geschickt wird, um den Fortschritt des aktuell gespielten MP3-Files mitzuteilen. Damit der MP3ReadNode dieses Event verschickt, muss ihm vorher ein *send_progress* Event mit einem “true” Value ge-

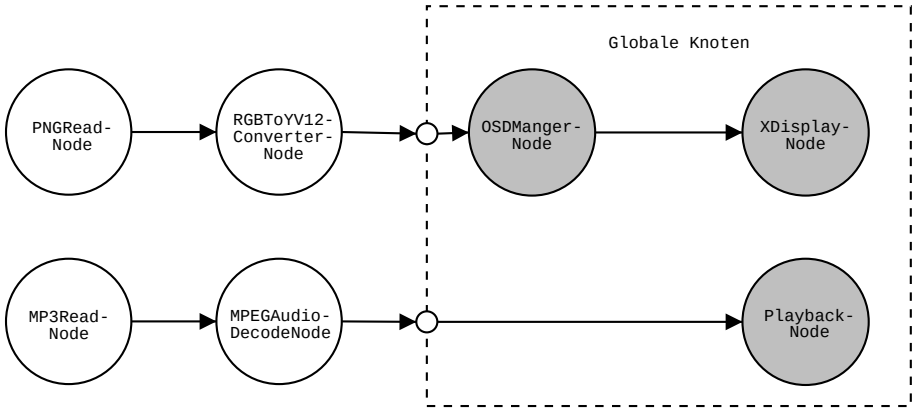

Abbildung 6.24: Aufbau eines NMM-Graphen für den MP3 Player

schickt werden. Der MP3State-Listener für das *progress* Event sorgt dafür, dass die Progressbar des OSDManagerNode aktualisiert wird.

Der Eintrag für den MP3State in der XML-Datei lautet:

```
<mp3play id="MP3Player" background="back.png"
         needvideo="no" needaudio="yes">
</mp3play>
```

Teile des MP3State wurden aus [65] entnommen.

6.3.7 Playlist

Die in Abschnitt 6.1.7 vorgestellte Playlist wird durch den gleichnamigen Zustand realisiert. Die grafischen Darstellung der drei Windows wird durch drei Listunits (Abschnitt 5.1.8) ermöglicht. Sie bieten die Möglichkeiten Filenamen, CD-Titel usw. in einer skrollbaren Liste anzuzeigen. Abhängig in welchem Window man sich befindet, werden entweder andere Windows verändert (Einträge hinzugefügt oder gelöscht) oder eine selektierte Datei abgespielt.

Im Playlist-Window befinden sich alle hinzugefügten Einträge, dabei erkennt man am Namen nicht unbedingt das Medium, von dem Daten abgespielt werden sollen. Ein Eintrag “Track01” könnte sowohl ein MP3 File auf Festplatte als auch ein Titel auf der eingelegten CD meinen. Beim Hinzufügen eines Eintrags in die Playliste wird deshalb eine URL generiert, die den Eintrag genau spezifiziert. Diese URL wird zusätzlich in einem TextView Widget (Abschnitt 5.1.6) gespeichert. Beim Beenden des Zustandes oder Verlassen der MMBox-Anwendung wird der Inhalt der Playlist in der Datei “mmbox.pls” im MMBox-Root-Verzeichnis abgespeichert. Startet man den Playlist Zustand, wird diese Datei wieder eingelesen.

Soll ein Eintrag abgespielt werden, so wird dessen URL an einen *GraphBuilder* [66] weitergeleitet. Der GraphBuilder erzeugt mit Hilfe der angegebenen URL einen CompositeNode (Abschnitt 3.2.2), der alle Knoten enthält, um die URL ab-

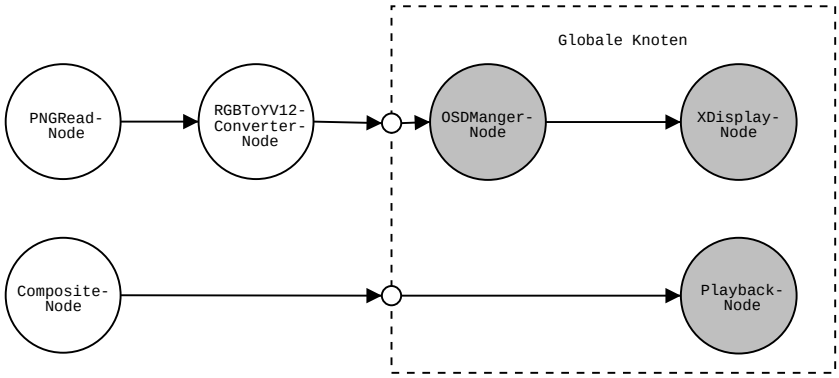

Abbildung 6.25: Aufbau eines NMM-Graphen für einen CompositeNode, der keinen Videoausgang besitzt.

zuspielen. Dabei sind die Ausgabeknoten nicht enthalten, d.h. der erzeugte CompositeNode kann direkt mit den globalen Knoten der MMBox Applikation verbunden werden. Der CompositeNode liefert einerseits die bekannten Knoten, um eine Audio-CD, MP3- oder MPEG Videos abzuspielen, andererseits können mit seiner Hilfe auch AVI-Dateien, die Videos im DivX Format [1] speichern, abgespielt werden. Mit Hilfe des Graphbuilders und CompositeNodes wird die Integration zum Abspielen neuer Dateiformate sehr vereinfacht. Weiterführende Informationen über den GraphBuilder können in [66] gefunden werden.

Um eine URL abzuspielen muss zuerst ein GraphBuilder erzeugt werden und die URL übergeben werden:

```
GraphBuilder gb;
gb.setURL( URL );
CompositeNode* cn = gb.createGraph();
```

Die `createGraph()` Methode liefert einen CompositeNode zurück; konnte kein CompositeNode für die angegebene URL erzeugt werden, so wird ein NULL-Zeiger zurückgeliefert. Ein CompositeNode besitzt abhängig von der übergebenen URL verschiedene Ausgänge. An die globalen Knoten kann jeweils ein Video- und ein Audioausgang verbunden werden. Die Ausgangsjacks des CompositeNode müssen daher überprüft werden, ob sie einen Video und/oder Audioausgang besitzen:

```
const list<const char*>* cn_outputs =
  cn->getOutputStreamTags();
```

Liefert eine Liste mit Bezeichnungen aller Ausgangsjacks. Geprüft werden muss auf Jacks mit den Namen “video” und “audio”. Abbildung 6.25 zeigt, wie der CompositeNode mit den globalen Knoten verbunden wird, wenn kein Videoausgang vorhanden ist. Analog wird der CompositeNode mit dem OSDManagerNode verbunden, wenn kein Audio aber ein Videoausgang vorhanden ist. Bei vorhande-

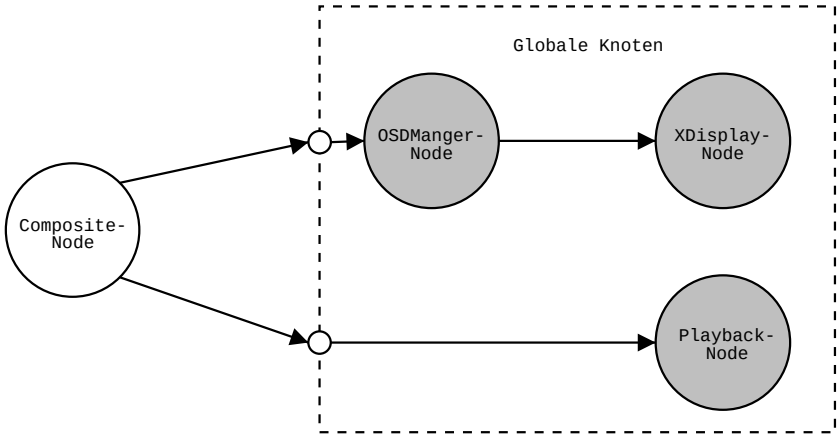

Abbildung 6.26: Aufbau eines NMM-Graphen für einen CompositeNode, der sowohl Audio- als auch ein Videoausgang besitzt.

nem Audio- und Videoausgang müssen beide mit den globalen Knoten verbunden werden (Abbildung 6.26).

Der GraphBuilder und der CompositeNode befinden sich noch in der Entwicklung [66]. Geplant ist, dass auch die globalen Knoten in einem CompositeNode untergebracht werden. Beim Verbinden zweier CompositeNodes sollen dann automatisch die “richtigen” Jacks miteinander verbunden werden.

Damit ein Playlist Zustand erstellt wird, muss das XML Dokument erweitert werden:

```
<playlist id="Playlist" background="back.png"
          needvideo="yes" needaudio="yes">
  <win id  = "Selection"
      x    = "50"
      y    = "320"
      width = "100"
      height = "200"

      win_active_color = "000 000 255 128"
      win_inactive_color = "128 128 128 128"

      sel_active_color = "128 255 000 255"
      sel_active_color2 = "255 128 000 255"
      sel_inactive_color = "128 128 128 255"></win>

  <win id  = "Track"
      x    = "170"
      y    = "320"
      width = "500"
      height = "200"
```

```
        win_active_color = "000 000 255 128"
        win_inactive_color = "128 128 128 128"

        sel_active_color = "128 255 000 255"
        sel_active_color2 = "255 128 000 255"
        sel_inactive_color = "128 128 128 255"></win>

    <win id  = "Playlist"
        x    = "50"
        y    = "50"
        width = "620"
        height = "250"

        win_active_color = "000 000 255 128"
        win_inactive_color = "128 128 128 128"

        sel_active_color = "128 255 000 255"
        sel_active_color2 = "255 128 000 255"
        sel_inactive_color = "128 128 128 255"></win>
  </playlist>
```

Zwischen dem “playlist” Tag müssen drei “win” Tags eingefügt werden, die die Position und Erscheinung der drei Windows beschreiben:

id Name des Windows, das konfiguriert werden soll. Dieses Attribut steht entweder auf “Selection”, “Track” oder “Playlist”.

x X-Koordinate der oberen linken Ecke des Windows.

y Y-Koordinate der oberen linken Ecke des Windows.

width Breite des Windows.

height Höhe des Windows.

win_active_color Farbe des Windows, wenn es aktive ist, d.h. wenn es den Eingabefocus besitzt. Der String muss vier Werte im Bereich von 0 bis 255, getrennt durch ein Leerzeichen, enthalten. Sie beschreiben den Farbwert im RGBA Format.

win_inactive_color Farbe des Windows, wenn es keinen Eingabefocus besitzt

sel_active_color Farbe des Selektionsbalken, wenn das Window aktiv ist

sel_active_color2 Alternative Farbe des Selektionsbalken bei aktivem Window. Diese Farbe wird zum Beispiel verwendet, um den Editmodus im Playlist-Window zu kennzeichnen.

sel_inactive_color Farbe des Selektionsbalken, wenn das Window keinen Eingabefocus besitzt.

6.3.8 TV Timer

Der in Abschnitt 6.1.6 vorgestellte TV Timer wird durch den gleichnamigen Zustand implementiert. Zur grafischen Interaktion wird die in Abschnitt 5.1.8 vorgestellte DoubleListUnit verwendet, die es ermöglicht, verschiedene Einträge in eine Liste einzufügen. Soll ein Timer hinzugefügt werden, so werden alle Werte der DoubleListUnit herangezogen, um folgende Struktur zu füllen:

```
enum status_e{ SLEEPING=0, RECORDING };
typedef struct {
  time_t    start_time;
  time_t    end_time;
  unsigned int channel;
  status_e  status;
} rec_entry_t;
```

Diese Struktur definiert genau einen Timer. In “start_time” und “end_time” ist die Uhrzeit für Beginn und Ende der Aufnahme gespeichert. “channel” ist die Nummer des aufzunehmenden Senders. “status” wird auf “SLEEPING” gesetzt, wenn die Aufnahme noch nicht gestartet wurde, der Timer also noch auf seine Aktivierung wartet. Der Status wird auf “RECORDING” gesetzt, wenn eine Aufnahme gestartet wurde.

Alle Timer werden in einer Liste gespeichert:

```
typedef map<unsigned int id, rec_entry_t*> rec_table_t;
rec_table_t rec_table;
```

Jeder Timer erhält beim Hinzufügen eine Nummer (ID), mit der er sich später identifizieren lässt. Bevor ein Timer in diese Liste aufgenommen wird, wird geprüft, ob es keine Überschneidungen mit bereits vorhandenen Timern gibt. Kann der Timer hinzugefügt werden, wird sein Status zunächst auf “SLEEPING” gesetzt und ein *TVTIMER* Event mit der ID des Timers zu einem `TimerEvents` Objekt hinzugefügt. Die `TimerEvents` Klasse sorgt dafür, dass das Event zur angegebenen Zeit weggeschickt wird (vgl. nächster Abschnitt). Der Zustand selbst registriert sich für ein *TVTIMER* Event. Empfängt der Zustand ein solches Event, so kann an Hand der ID festgestellt werden, welcher Timer gestartet oder gestoppt werden soll. Befindet sich der Status auf “SLEEPING”, so wird die Aufnahme gestartet und der Status wird auf “RECORDING” gesetzt. Ist der Status bereits auf “RECORDING”, dann wird die Aufnahme gestoppt und der Timer aus der Liste gelöscht (Abbildung 6.27).

TimerEvents

Die Klasse `TimerEvents` ist vergleichbar mit Producern aus Abschnitt 5.3. Einem `TimerEvents`-Objekt können beliebig viele Events hinzugefügt werden, die mit Hilfe eines Eventdispatchers (Abschnitt 3.3) zu bestimmten Zeiten ausgeführt werden.

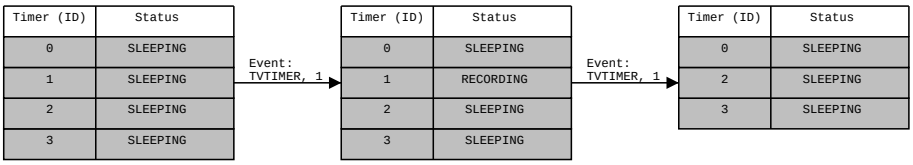

Abbildung 6.27: Die Timerliste besteht aus 4 programmierten Aufnahmen, die alle den Staus “SLEEPING” haben. Beim Empfang eines *TVTIMER* Events, wird die angegebene Aufnahme gestartet und der Status auf “RECORDING” gesetzt. Beim nochmaligen Empfang des Events, wird die Aufnahme gestoppt und aus der Liste gelöscht.

void connectTo(EventDispatcher*) Übergibt einen Eventdispatcher, der die zeitgesteuerten Events verarbeiten soll

void start() Startet die Verarbeitung zeitgesteuerte Events

void stop() Stoppt die Verarbeitung zeitgesteuerte Events

void registerTimer(int Sekunde, int Minute, int Stunde, int Tag, int Monat, int Jahr, Event* Event) Registriert ein Event, das zur angegebenen Zeit verschickt werden soll

registerTimer(time_t t, Event* Event) Registriert ein Event, das zur angegebenen Zeit verschickt werden soll, dabei wird die absolute Zeitinformation mit Hilfe des `time_t` Types der Standardbibliothek übergeben

Alle Zeiten mit ihren zugehörigen Events werden in einer Prioritätenschlange abgelegt. Somit wird gewährleistet, dass die Zeitpunkte, mit dem frühsten Zeitpunkt beginnend, sortiert sind. Die Abarbeitung der Events wird dadurch erleichtert. Es muss immer nur auf den Zeitpunkt gewartet werden, der in der Prioritätenschlang als erster aufgeführt ist. Ist der Zeitpunkt eingetreten, wird das Event verschickt und auf den Zeitpunkt des nächste Eintrags der Schlange gewartet.

Der Event-Verarbeitungsteil der `TimerEvents` Klasse läuft in einem eigenen Thread. Mit Hilfe der

```
pthread_cond_timedwait(pthread_cond_t Conditional,
  ThreadMutex Mutex, timespec Zeitpunkt)
```

Funktion der PThread [13] Bibliothek wird der Thread solange blockiert bis der angegebene Zeitpunkt erreicht wird oder die angegebene Conditional-Variable benachrichtigt wird. Die Conditional-Variable wird benötigt, wenn ein neues Event hinzugefügt wurde, das zu einem früheren Zeitpunkt ausgeführt werden soll als das Event, auf das gerade gewartet wird. Dann muss das Warten unterbrochen werden und auf den früheren Zeitpunkt gewartet werden (Abbildung 6.28).

Der Eintrag für den TvTimer in der XML-Datei lautet:

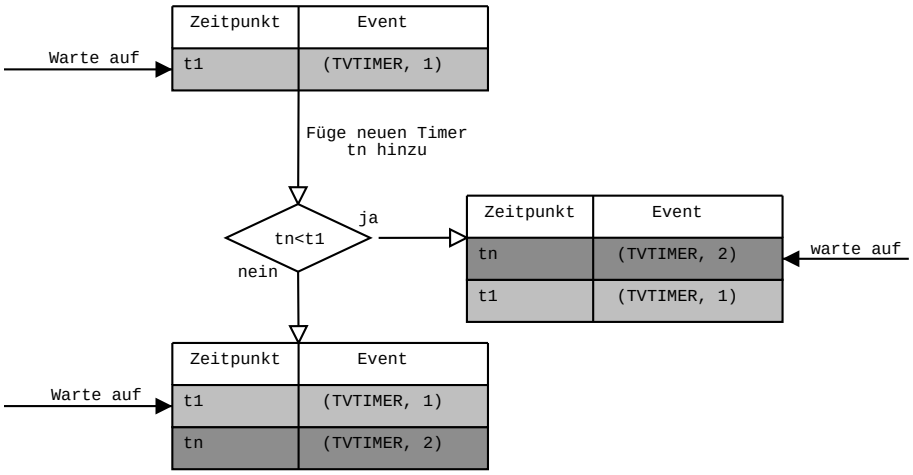

Abbildung 6.28: Wird ein neues Event registriert (hinzugefügt), muss das gerade wartende Event nur dann unterbrochen werden, wenn der Zeitpunkt des neuen Events vor dem des wartenden Events liegt.

```
<tvtimer id="TV-Timer" background="back.png"
         needvideo="no" needaudio="no">
</tvtimer>
```

Alle Timer werden beim Verlassen des Zustandes oder der Anwendung in der Datei “timer.xml” gespeichert. Beim Starten der Anwendung wird diese Datei wieder eingelesen und alle darin enthaltenen Timer vom Zustand übernommen. Folgendes XML Dokument beinhaltet 4 gespeicherte Timer:

```
<?xml version="1.0"?>
<tvtimers>
  <timer start="2003-05-20␣10:12" stop="2003-05-20␣10:13"
     channel="3"></timer>
  <timer start="2004-01-20␣10:14" stop="2004-01-20␣10:15"
     channel="1"></timer>
  <timer start="2003-10-20␣10:14" stop="2003-10-20␣10:15"
     channel="1"></timer>
  <timer start="2003-01-25␣10:15" stop="2003-01-25␣10:16"
     channel="3"></timer>
</tvtimers>
```

Zwischen den “tvtimers” Tags befindet sich für jede programmierte Sendung ein “timer” Tag. Die Attribute “start” und “stop” geben die Start- und Stopzeit an. Die Zeit wird im Format “Jahr-Monat-Tag Stunde:Minute” gespeichert. Das “channel” Attribut beinhaltet die Kanalnummer.

Der TvTimer Zustand dient nur zum Programmieren und Starten zeitgesteuerter Aufnahmen. Die eigentlichen Aufnahmen werden durch den DvbRecorderState realisiert. Der TvTimer Zustand initialisiert einen DvbRecorderState und

übergibt ihm eine Nachricht (Abschnitt 6.2.5) mit der Kanalnummer und Kanalnamen (Nummer:Name), der aufgenommen werden soll. Die Aufnahme startet sofort, ein eventuell laufender TvViewer Zustand wird beendet. Der NMM-Graph besteht aus einem DVBReadNode, der zum gewählten Kanal umschaltet und das TV-Programm als MPEG-Strom verschickt und einem RawWriteNode, der den MPEG-Strom auf Festplatte speichert (Abbildung 6.29). Das gewählte Programm wird im MMBox Rootverzeichnis in einem Unterordner angelegt, der in der MMBox Konfigurationsdatei unter “tvdestinationdirectory” angegeben wurde. Der Dateiname ergibt sich aus Kanalname und Zeitinformation. Beispiel: “ARD 2002-12-30 20:15.mpg”.

Auch der DvbRecordState muss im XML File angegeben werden:

```
<dvbrecord id="DVB-Recorder"
           needvideo="no" needaudio="no">
</dvbrecord>
```

Wegen der einfacheren Implementation wurde für die Aufnahme ein neuer Zustand (DvbRecordState) erstellt. Nachteil dieser Methode ist, dass der DvbState bei aktivem DvbRecordState nicht mehr gestartet werden kann. Eine bessere Implementation wäre die Erweiterung des DvbState um die Fähigkeiten des DvbRecordState, damit der DvbState die Aufnahme übernehmen kann.

6.3.9 ConfigState

Der ConfigState bietet die Möglichkeit, das Verhalten der Zustände zu verändern. In diesem Zustand können verschiedene Werte verändert werden. Damit Zustände auf die gesetzten Werte zugreifen können, besitzt die MMBox Applikation ein `MMBoxConfig` Objekt, das in [65] entwickelt wurde. Einem Schlüssel kann dabei genau ein Wert zugeordnet werden.

```
void setValue( string, string );
bool getValue(const string, string&);
bool getValue(const string, bool&);
bool getValue(const string, int&);
```

Mit `setValue` kann man einem Schlüssel einen Wert zuweisen. Die Methode `getValue` liefert den Wert eines Schlüsseln zurück. Dabei wird auch der Typ

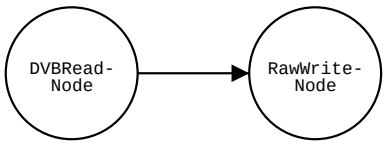

Abbildung 6.29: Realisierung einer TV-Aufnahme: Der DVBReadNode verschickt das TV-Programm als MPEG Strom, der RawWriteNode speichert diesen Strom auf Festplatte.

berücksichtigt. Angenommen in der Konfigurationsdatei stehen folgenden Zeilen:

```
useac3 = "yes"
cddb   = "false"
```

Dann liefert ein Aufruf von `getValue("useac3", value)` den Wert **`true`** zurück, wenn `value` eine bool-Variable ist. Die Strings “true”, “yes” und “1” liefern einen **`true`** Wert, alle anderen ein **`false`** Wert.

Der ConfigState interagiert mit Hilfe einer DoubleListUnit (Abschnitt 5.1.8) mit dem Benutzer. Sie bietet die grafische Auswahlmöglichkeit der einzelnen Optionen (Schlüssel) und die Schlüsselwerte können verändert werden. Durch einen Druck auf die Play-Taste werden die Einstellungen der DoubleListUnit in das Konfigurationsobjekt übernommen und können somit von den Zuständen abgefragt werden.

Welche Einträge in dem Konfigurationsfenster (DoubleListUnit) erscheinen sollen, wird durch das XML Dokument festgelegt:

```
<config id="Configuration" background="back.png"
        needvideo="no" needaudio="no">
  <option id = "AC3 Passthrough"
        short = "useac3">no yes</option>
  <option id = "Use CDDB"
        short = "cddb">yes no</option>
  <option id = "MP3 Enc. Bitrate"
        short = "grabmp3bitrate">128 192 64</option>
  <option id = "Use hardware MPEG decoder"
        short = "hard_mpeg">yes no</option>
</config>
```

Zwischen den “config” Tags werden alle Optionen mittels “option” Tag eingefügt. Es hat wiederum weitere Attribute, die die Option näher bestimmen:

id Bezeichnet den Text, der im Auswahlfenster erscheinen soll

short Gibt den Schlüssel an, über den man auf den Wert der Option zugreifen kann

Zwischen dem Beginn und Ende eines “option” Tags stehen alle Werte, die der Option zugewiesen werden können. Sie werden durch ein Leerzeichen getrennt. Der erste Wert ist immer der Standardwert.

6.3.10 TaskMgrState

Mit dem TaskMgrState kann zwischen aktiven Zuständen gewechselt werden und sie können beendet werden (Abschnitt 6.1.9).

Die MMBox Applikation hat eine Liste mit allen aktiven Zuständen. Sie werden mit einer Listunit (Abschnitt 5.1.8) in einem Fenster ausgegeben:

```
/* Trage alle aktiven Zustände in eine Liste ein */
void TaskMgrState::activeTaskToList() {
```

```
  const MMBoxApplication::mmbox_state_list_t* states;
  MMBoxApplication::mmbox_state_list_t::const_iterator i;

  /* Hole alle aktiven Zustände */
  states = (MMBoxApplication*) app->getActiveStates();

  /* Speichere die Namen der Zustände in einer Listunit */
  for( i=states->begin(); i!=states->end(); i++ ){
    MMBoxState* state = *i;
    (ListUnit*) list->getTextWidget()->
             addLine( state->getID() );
  }
}
```

Die Listunit speichert in jeder Zeile den Namen eines aktiven Zustands. Je nach Benutzereingabe wird bei einem selektierten Listeneintrag zum selektierten Zustand gewechselt:

```
/* Wechsle zum selektierten Zustand */
Result TaskMgrState::Return() {
  /* ermittle die selektierte Zeile */
  unsigned int sel = (ListUnit*) list->getTextWidget()->
     getSelectedLine();

  /* Hole den Namen des selektierten Zustandes */
  string id = (ListUnit*) list->getTextWidget()->
     getLineContent( sel );

  /* wechsle zum angegebenen Zustand */
  changeState( id );

  return(SUCCESS);
}
```

Oder der selektierte Zustand wird beendet:

```
Result TaskMgrState::specialKey01() {
  /* ermittle die selektierte Zeile */
  unsigned int sel = list->getTextWidget()->getSelectedLine
     ();

  /* Hole den Namen des selektierten Zustandes */
  string id = list->getTextWidget()->getLineContent( sel );

  /* beende selektierten Zustand, wenn es
     nicht der TaskMgrState selbst ist */
  if ( id != getID() ){
    app->getState(id)->kill();
  }

  return(SUCCESS);
```

```
}
```

Der Eintrag für den TaskMgrState in der XML-Datei lautet:

```
<taskmgr id="TaskManager" background="back.png"
         needvideo="no" needaudio="no">
</taskmgr>
```

6.3.11 Erweiterungen

In diesem letzten Abschnitt soll an Hand eines Beispiels gezeigt werden, welche Möglichkeiten bestehen, die Multimedia-Box Anwendung zu erweitern. Ziel ist die Entwicklung einer Slideshow, die alle Bilder, die sich in einem Verzeichnis befinden, nacheinander anzeigt. Die Slideshow soll als neuer Zustand in die Multimedia-Box integriert werden. Dabei sollen die Bilder nach einer bestimmten Zeit oder durch einen Tastendruck wechseln. Die Zeitinformation soll im XML-Dokument verändert werden können.

Das Beispiel beschränkt sich auf die wesentliche Aufgabe, also die Implementierung bestimmter Methoden des Zustandes und die Integration des Zustands in die Anwendung. Erweiterte Funktionalität, wie die Auswahl des Verzeichnisses, in dem sich die Bilder befinden, werden nicht implementiert.

Um einen neuen Zustand zu integrieren, müssen verschiedene Schritte durchgeführt werden:

Schritt 1: Auswahl der NMM-Knoten

Eine Slideshow kann durch einen geeigneten NMM-Flussgraphen realisiert werden. Der Flussgraph besteht aus 2 Knoten:

- PNGReadNode (vgl. Abschnitt 5.2.1.4)
- RGBtoYV12ConverterNode (vgl. Abschnitt 5.2.2.7)

Der Flussgraph ist wie in Abbildung 6.30 aufgebaut. Unterstützt werden Bilder im PNG-Format. Der PNGReadNode liest die Bilder von der Festplatte und schickt sie an den RGBtoYV12ConverterNode. Dieser Knoten wird benötigt, da der globale OSDManagerNode auf das YV12-Format eingestellt ist.

Schritt 2: Implementierung des Zustandes

Der Zustand soll den Namen SlideshowState tragen. Er wird, wie jeder Zustand, von der Klasse `MMBoxState` abgeleitet:

```
class SlideshowState : public MMBoxState {

public:
  SlideshowState( string, MMBoxApplication* );
  ~SlideshowState();
```

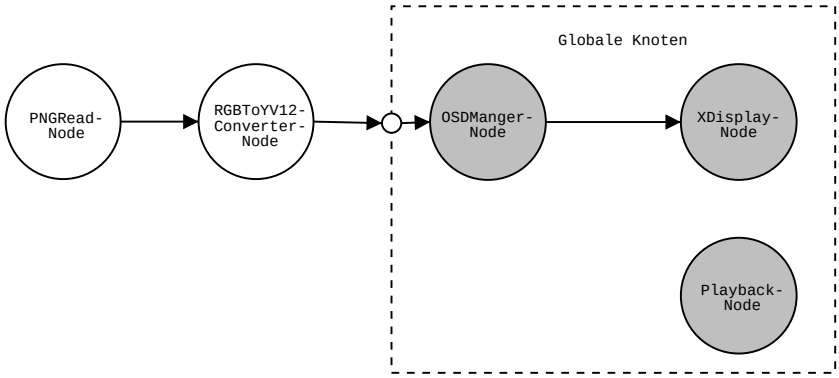

Abbildung 6.30: Flussgraph der Slideshow. Bilder werden vom PNGReadNode von der Festplatte gelesen, danach ins YV12 Farbformat konvertiert, an den OSD-ManagerNode geleitet und danach vom XDisplayNode angezeigt.

```
  /* Navigationstaste : runter */
  Result Down();

  /* Navigationstaste : Back */
  Result Back();

  Result doCreate( string );
  Result doKill();

  /**
    * setzt die Zeit (in Sekunden), die zwischen dem Wechsel
    * zwischen 2 Bildern gewartet werden soll
    */
  setDuration( unsigned int s ){
   seconds_to_wait = s;
  }

protected:
  /* Zeit (Sekunden), die zwischen dem Wechsel
     zwischen 2 Bildern gewartet werden soll */
  unsigned int  seconds_to_wait;

  /* Zeiger auf den OSDManagerNode */
  OSDManagerNode* osd_node;

  /* Zeiger auf den PNGReadNode */
  PNGReadNode*  pngread_node;

  /* Zeiger auf den RGBtoYV12ConverterNode */
  RGBtoYV12ConverterNode* converter_node;
```

```
  /* Dispatcher, der TimerEvents verarbeitet */
  EventDispatcher* dispatcher;

  /* Producer, der TimerEvents erzeugt */
  TimerEvents*  timerevents;

  /* MMBox Rootpath (aus diesem Verzeichnis
     werden die Bilder entnommen) */
  string        rootpath;

  /* Dateinamen der PNG-Datei */
  string        filename;
};
```

Im Konstruktor wird der Rootpath gesetzt und der Producer für die Timer-Events mit dem Dispatcher verbunden:

```
SlideshowState::SlideshowState( string id, MMBoxApplication
    * _app, BitmapReader*) : MMBoxState( id, _app ){
  /* Ermittle Rootpath aus der MMBox-Konfigurationsdatei */
  MMBoxConfig* config = app->getMMBoxConfig();
  if (!(config->getValue("rootpath", rootpath))) {
   rootpath="/";
  }

  /* Fordere globalen OSDManagerNode an */
  osd_node = app->getOsdNode();

  /* Fordere Event-Dispatcher an */
  dispatcher = app->getDispatcher();

  /* Fordere Producer für Timer-Events an */
  timerevents = app->getTimerProducer();

  /* Verbinde Producer mit Dispatcher */
  if ( dispatcher )
    dispatcher->registerEvent( "SLIDETIMER", new TEDObject0<
        SlideshowState>( this, &SlideshowState::down ));
}
```

Die `doCreate`-Methode wird aufgerufen, wenn der Zustand erstellt wird:

```
Result SlideshowState::doCreate( string ){
  /* Erstelle Knoten */
  pngread_node = new PNGReadNode();
  converter_node = new RGBtoYV12ConverterNode();

  /* Initialisiere Knoten */
  pngread_node->init();
  converter_node->init();
```

```
  /* Setze Dateinamen */
  pngread_node->sendSetEvent(Event("usefile",
        new TInValue<string>( filename )));

  /* Verbinde die Knoten */
  pngread_node -> connectOutputTo( converter_node,
                formatVideoRGBA32);
  converter_node -> connectOutputTo( osd_node, "stream1",
                formatVideoYV12);
  osd_node    -> connectOutputTo( display_node,
                formatVideoYV12);

  /* Aktiviere Knoten */
  pngread_node -> activate();
  converter_node -> activate();
  osd_node -> activate();

  /* Starte Knoten */
  pngread_node -> start();
  converter_node -> start();
  osd_node -> start();

  /* Schicke SLIDETIMER-Event nach "seconds_to_wait"-Sekunden weg */
  Event* e1 = new Event( "SLIDETIMER", 1 );
  timerevents->registerTimer( time(0)+seconds_to_wait, e1 );

  return( SUCCESS );
}
```

Wenn der Zustand beendet wird, wird folgende Methode aufgerufen:

```
Result SlideshowState::doKill() {
  /* Stoppe Knoten */
  pngread_node -> stop();
  converter_node -> stop();
  osd_node -> stop();

  /* Deaktiviere Knoten */
  pngread_node -> deactivate();
  converter_node -> deactivate();
  osd_node -> deactivate();

  /* Trenne Knotenverbindungen */
  converter_node -> disconnectInput();
  osd_node -> disconnectInput();

  /* Lösche Knoten */
  delete pngread_node:
  delete converter_node;
```

```
  return( SUCCESS );
}
```

Wird die Back-Taste gedrückt, so wird zum letzten laufenden Zustand oder ins Hauptmenü gewechselt:

```
Result SlideshowState::Back() {
  return( goToLastState() );
}
```

Sobald die Zeit abgelaufen ist oder die Navigationstaste “runter” gedrückt wird, wird das nächste Bild angezeigt:

```
Result SlideshowState::down(){
  /* Hole Namen der PNG-Datei*/
  filename = ...

  /* Gib PNG-Dateinamen an PNGReadNode weiter */
  pngread_node->sendSetEvent(Event("usefile", new TInValue<
      string>( filename )));

  /* Schicke SLIDETIMER-Event nach "seconds_to_wait"-Sekunden weg */
  Event* e1 = new Event( "SLIDETIMER", 1 );
  timerevents->registerTimer( time(0)+seconds_to_wait, e1 );
}
```

Schritt 3: Integration des Zustandes

Nachdem der Zustand implementiert wurde, muss er in die Multimedia-Box integriert werden. Der Eintrag für den SlideshowState kann in der XML-Datei wie folgt aussehen:

```
<slideshow id="Slideshow" duration="5"
          needvideo="yes" needaudio="no">
</slideshow>
```

Das Attribut “duration” gibt dabei an, wie lange ein Bild angezeigt werden soll. Der XML-Parser erstellt dabei die Instanzen der Zustände. Er muss so erweitert werden, dass der neue Zustand erstellt wird, wenn der entsprechende Eintrag dafür in der XML Datei vorhanden ist:

```
bool XmlParser::parseSlideshow( MMBoxApplication* app ){

  /* Hole Liste mit allen <slideshow> Tags */
  XMLNodeList slideshow_list;
  slideshow_list = rootnode->children( "slideshow" );

  XMLNodeIterator iter, stop;

  iter = slideshow_list.begin();
```

```
  stop = slideshow_list.end();

  /* Verarbeite alle <slideshow>-Tags */
  while (iter != stop) {
    XMLNode* node = *iter;
    iter++;

    XMLProperty* prop;

    /* Ermittle "id" Attribute */
    string str_pid;
    prop = node->property( "id" );
    if ( prop )str_pid = prop->value();

    /* Erstelle Instanz des Zustandes */
    SlideshowState* slideshowstate_state =
      new SlideshowState( str_pid, app );

    /* Suche "duration" Attribut und setze Wert */
    prop = node->property( "duration" );
    if ( prop )slideshowstate_state->
       setDuration( atoi( prop->value().c_str() );

    /* Interpretiere "needaudio" und "needvideo" Attribute */
    parseStdState( node, slideshowstate_state );

    /* Füge der Applikation den Zustand hinzu */
    state_list.push_back( slideshowstate_state );

  }

  return( true );
}
```

Der neue Zustand kann erst aktiviert werden, wenn auch ein Menüeintrag vorhanden ist (vgl. Abschnitt 6.3.1).

Kapitel 7

Zusammenfassung und Ausblicke

7.1 Erzielte Ergebnisse

In dieser Arbeit wurde eine Multimedia-Box entwickelt, deren Entwicklung in einem Fortgeschrittenen-Praktikum [65] begonnen hat. Die erreichten Ziele dieser Arbeit sind:

- Die Multimedia-Box wurde aus Standard-PC Komponenten aufgebaut. Das Gehäuse hat in etwa die Form eines Videorekorders und lässt sich somit leicht ins Wohnzimmer integrieren. Damit die Laufgeräusche des PCs nicht stören, wurden besonders leise CPU- und Netzteillüfter eingebaut.

  Verschiedene Erweiterungskarten wurden integriert. Eine digitale Satellitenkarte zum Empfang von TV-Sendungen, eine Grafikkarte mit TV-Ausgang, damit die Multimedia-Box an einen Fernseher angeschlossen werden kann und eine Soundkarte, die einen digitalen Audioausgang besitzt und sich somit an einen digitalen Verstärker anschließen lässt.

  Ein integriertes DVD-Laufwerk bietet die Möglichkeit sowohl DVDs als auch CDs zu lesen. Ein Infrarotempfänger sorgt dafür, dass die Multimedia-Box mit einer Fernbedienung gesteuert werden kann. Mit dem vorhandenen LC-Display lassen sich zusätzliche Textinformationen anzeigen.

- Die grafische Benutzerschnittstelle kann mit dem entwickelten Toolkit realisiert werden. Dieses Toolkit basiert auf bekannten Mechanismen, die zur Darstellung der Oberfläche Windows und Widgets verwenden [41]. Windows und Widgets können in ihrer Größe angepasst werden.

  Hierarchische Menüs können erstellt werden, deren Menüpunkte aus verschiedenen Grafiken bestehen können. Die Menüknöpfe können je nach Selektionszustand ihr Aussehen verändern.

  Fortschrittanzeigen können in eine Anwendung integriert werden, um den Status eines Vorgangs anzuzeigen. Listboxen, in denen Texteinträge aufgelistet werden, können vom Benutzer selektiert und verändert werden. Ausge-

wählte Listeneinträge werden durch Balken, die verschiedene Farben besitzen können, hervorgehoben.

Das Decorator-Design-Pattern [22] wurde verwendet, um den grafischen Objekten dynamisch Attribute wie Rahmen und verschiedenfarbige Hintergrundebenen hinzuzufügen.

Es wurde ein Mechanismus implementiert, mit dem Zeichensätze mit Hilfe einer XML Beschreibung aus einer Grafikdatei ausgelesen werden können.

- Die Software baut auf der Netzwerk-Integrierten Multimedia Middleware (NMM) auf, die am Lehrstuhl für Computergrafik entwickelt wird (vgl. Kapitel 3). Zur Realisierung der Funktionen wurden NMM-Flussgraphen aufgebaut, wobei neue NMM-Knoten entwickelt, vorhandene Knoten erweitert oder optimiert und Knoten unverändert übernommen wurden [65]. Folgende Funktionen werden unterstützt:
  - Wiedergabe von Video-DVDs, dabei werden auch vorhandenen Menüs angezeigt, in denen der Benutzer je nach DVD Sprache, Untertitel und Bonusmaterial auswählen kann.
  - TV-Sendungen können empfangen und wiedergegeben werden, wobei auch Timeshifting-Funktionen unterstützt werden, die in [65] entwickelt wurden. TV-Programme können programmiert und zeitgesteuert aufgenommen werden.
  - Neben dem Abspielen von Audio-CDs ist auch das “Grabben” möglich, d.h. die Audio-CD wird ausgelesen, in das platzsparendere MP3-Audioformat konvertiert und danach auf Festplatte geschrieben, dabei können Interpret und Titelinformationen aus einer Internetdatenbank abgefragt werden. Diese Funktion wurde in [65] implementiert und in dieser Arbeit so erweitert, dass Zeit- und Fortschrittsinformationen während der Wiedergabe angezeigt werden.
  - Die Playlist bietet die Möglichkeit verschiedene Dateien, die unterschiedliche Formate haben abzuspielen, wie z.B. MP3, AVI, MPEG-Dateien oder CD-Titel. Diese Dateien können in eine Liste aufgenommen werden, in der sie der Reihe nach abgespielt werden. Die Listeneinträge können gelöscht und verschoben werden. Verschiedenfarbige Balken zeigen die aktuell gespielte Datei und die Benutzerinteraktion an.
  - Im Taskmanager werden alle laufenden Funktionen aufgelistet. Funktionen können in der Liste selektiert werden, wobei zur Funktion hingeschaltet werden kann oder die Funktion beendet werden kann.
- Dem Anwendungs-Framework liegt das State-Design-Pattern [22] zugrunde. Abhängig vom Zustand der Anwendung, ändert sich auch ihr Verhalten. Zustandsänderungen werden durch Starten und Stoppen von Funktionen oder durch Hin- und Herschalten zwischen Funktionen verursacht.

Zustände können auch Menüs erstellen, in denen navigiert werden kann und Funktionen gesteuert werden können.

Bei der Implementierung der Funktionen wird das grafische Toolkit verwendet. Funktionen können parallel ausgeführt werden. Sie werden durch ein XML-Dokument konfiguriert. Auch die Struktur der Anwendung selbst wird in einem XML-Dokument beschrieben. Dieses Dokument beschreibt, ob die Funktion ein Geräte wie Grafik- oder Soundkarte benötigt. Abhängig davon wird bei einem Zustandswechsel entschieden, ob eine Funktion beendet werden soll, oder ob die parallele Ausführung möglich ist.

Benutzereingaben werden, abhängig vom Zustand der Anwendung, an eine Funktion weitergeleitet. Die Benutzereingaben selbst werden durch Producer realisiert. Producer abstrahieren die eigentlichen Geräte, mit denen Benutzereingaben durchgeführt werden. Es können beliebig viele Producer mit der Anwendung verbunden werden. Momentan existieren Producer für Tastatur und Infrarot-Fernbedienung.

7.2 Ausblick

Im folgenden werden einige Ideen vorgestellt, wie die Multimedia-Box erweitert werden kann bzw. wo sie noch eingesetzt werden kann.

7.2.1 Neue Oberfläche

Die Multimedia-Box kann durch Austausch der Grafiken für Menüpunkte und Hintergrund an persönliche Vorlieben angepasst werden. Zahlreiche neue Oberflächen können generiert werden. Als Erweiterung könnten verschiedene *Themes* integriert werden, d.h. der Benutzer kann zwischen verschiedenen vorgefertigten Hintergrund und Menügrafiken auswählen während die Anwendung läuft. Vorstellbar ist auch, dass der Benutzer interaktiv Farben und Position der Fenster bestimmen kann.

Das Aussehen der Anwendung wird in einer XML Datei festgehalten, denkbar wäre auch die Entwicklung eines grafischen Toolkits, mit dem man diese Datei automatisch erzeugen könnte, d.h. Knöpfe sollten sich damit leicht erstellen und mit der Maus positionieren lassen.

Der statische Hintergrund könnte durch Videosequenzen ergänzt werden. Auch die Knöpfe sollten sich animieren lassen. Möglich wäre z.B. die Integration von Flash-Animationen [50] oder Szenenbeschreibungen im Scalable Vector Graphics (SVG) Format [83]. Damit können zweidimensionale Vectorszenen beschrieben werden, die animiert werden können.

7.2.2 Neue Funktionen

Eine Möglichkeit die Multimedia-Box mit neuer Funktionalität auszustatten, ist die Entwicklung neuer NMM-Knoten. Noch nicht vorhandene Video-Codecs können durch neue NMM-Knoten integriert werden. Besonders AVI-Dateien können viele unterschiedliche Video-Codecs enthalten. Interessant wären auch Knoten, die verschiedene Audio- oder Videoformate in andere (platzsparendere) Formate umwandeln können. Durch geeignete Flussgraphen könnten so *Transcoder* aufgebaut werden.

Spielt die Multimedia-Box reine Audiodaten ab, so sieht man die Abspielzeit und einen Fortschrittsbalken. Hier könnten verschiedene Visualisierungen entwickelt werden, wie etwa ein Oszilloskop oder ein Spektrum-Analyser, die die Audiodaten grafisch darstellen.

Neue Funktionen können auch durch die Entwicklung neuer Zustände realisiert werden. Vorstellbar sind hier verschiedene Spiele oder ein Web-Browser.

Erweitert man die Multimedia-Box um zusätzliche Hardware, wie z.B. einem CD oder DVD-Brenner, so muss auch hierfür ein Zustand entwickelt werden, der beispielsweise die Möglichkeit bietet verschiedene Audio Titel zusammenzustellen und auf CD zu brennen.

7.2.3 Ressourcen-Management

Die Multimedia-Box kann verschiedene Funktionen gleichzeitig ausführen. Die Anforderungen der Funktionen können dabei unterschiedlich sein. Einige Aufgaben müssen in “Echtzeit” ausgeführt werden, wie beispielsweise die Wiedergabe einer TV-Sendung. Andere Funktionen hingegen brauchen nicht unbedingt in “Echtzeit” ausgeführt zu werden, wie z.B. ein Enkoder, der eine Audio-CD nach MP3 kodiert. Werden beide Funktionen gleichzeitig ausgeführt, so könnte von einem Ressource-Manager sichergestellt werden, dass der MP3 Enkoder nur so viel CPU-Zeit bekommt, damit die TV-Sendung noch in “Echtzeit” wiedergegeben werden kann.

7.2.4 Verteilte Anwendungen

Mehrere Multimedia-Boxen könnten miteinander vernetzt werden, damit sie sich Ressourcen teilen. Rechenintensive Aufgaben könnten auf mehrere Boxen verteilt werden. Vorstellbar wäre auch, dass sich MMBoxen von anderen MMBoxen Geräte anfordern können, d.h. eine Multimedia-Box ohne TV-Karte könnte sich von einer anderen Box mit TV-Karte diese anfordern, damit diese dann die empfangenen Daten an die Box weiterleitet. Diese Erweiterung befindet sich schon in der Entwicklung [57, 51].

Interessant wäre dieser Mechanismus auch für mobile Geräte, d.h. mobile Geräte wie z.B. ein IPAQ könnten das Fernsehprogramm über eine MMBox empfangen. Dabei liefert die Box nicht die komprimierten MPEG Daten, da diese auf den schwächeren Prozessoren der mobilen Geräte nicht dekodiert werden können,

sondern die MMBox dekodiert die MPEG Daten, verringert die Bildauflösung und komprimiert die Bilder so, dass sie von einem mobilen Gerät dekodiert werden können. Teilweise ist dieses Szenario schon in [66] realisiert.

7.2.5 Benutzerinteraktion

Neben Tastatur und Infrarotfernbedienung wären auch andere Eingabegeräte vorstellbar. Hier könnte man sich wieder mobile Geräte vorstellen wie einen IPAQ oder auch Handies. Interessant wäre auch die Interaktion über Spracherkennung oder Gestik.

Anhang A

Tools und Treiber

A.1 TV-Out bei NVidia Grafikkarten

Der TV-Ausgang der NVidia Grafikkarten kann mit dem Tool NVTV [19] aktiviert werden. Es können zahlreiche Parameter eingestellt werden, um das Bild optimal für die Ausgabe auf einem Fernseher anzupassen. Neben der Auflösung kann auch der Overscanmodus aktiviert werden, der die schwarzen Rahmen um das Videobild entfernt. Das Tool kann mit Hilfe einer grafischen Oberfläche (Abbildung A.1) oder per Kommandozeilenoptionen konfiguriert werden. Für die TV-Ausgabe sollte eine Auflösung von 768x576 Pixeln im Overscanmodus (Large) benutzt werden:

```
nvtv -t -r 768,576 -s Large
```

A.2 Soundblaster Live

Die Soundblaster Live Soundkarte verfügt über einen digitalen Audioausgang. Damit dieser Ausgang unter Linux aktiviert werden kann, muss ein spezieller Treiber installiert werden (siehe [17]).

Nachdem der Treiber entpackt wurde, muss die Konfigurationsdatei “configfile” angepasst werden. Der Eintrag *KERNEL_SOURCE* muss den Pfad zu den Kernelsourcen beinhalten (meist /usr/src/linux). Danach kann mit *make* der Kompiliervorgang gestartet werden. Dann muss als Root *make install* ausgeführt werden. Im Verzeichnis “/lib/modules/2.4.xx/kernel/drivers/sound/emu10k1/” sollte nun eine Datei mit dem Namen “emu10k1.o” liegen.

Bei einem Debian-Linuxsystem muss die Datei “/etc/modutils/sound” wie folgt bearbeitet werden:

```
alias char-major-14 soundcore
alias char-major-116 snd
options snd snd_major=116 snd_cards_limit=1
alias snd-card-0 emu10k1
alias sound-slot-0 snd-card-0
alias sound-service-0-0 snd-mixer-oss
alias sound-service-0-1 snd-seq-oss
```

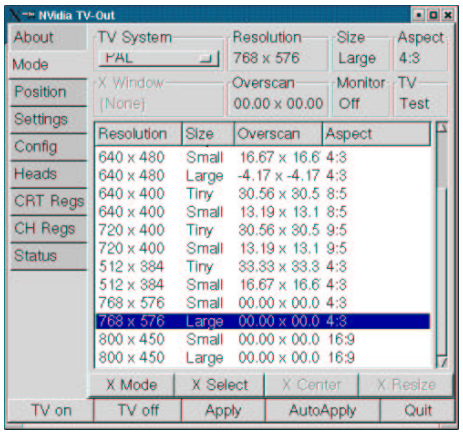

Abbildung A.1: Das NVTV Tool erlaubt die Aktivierung des TV-Ausgangs von NVidia Grafikkarten in fast allen Auflösungen und im Overscan Modus.

```
alias sound-service-0-3 snd-pcm-oss
alias sound-service-0-8 snd-seq-oss
alias sound-service-0-12 snd-pcm-oss
```

Danach muss der Befehl *update-modules* aufgerufen werden. Die Konfigurationsdatei “emu10k1.conf” befindet sich in “/usr/local/etc”, in der man die verschiedenen Ein- und Ausgänge der Soundblasterkarte aktivieren oder deaktivieren kann. Das Scriptfile “emu-script” liest diese Datei ein und verändert die Register der Soundkarte entsprechend. Dem Scriptfile können beim Starten Kommandozeilenparameter übergeben werden, die die Variablen der Konfigurationsdatei überschreiben (siehe emu-script für weitere Informationen).

Der Digitalausgang wird durch Setzen von *USE_DIGITAL_OUTPUT=yes* innerhalb des Scriptfiles aktiviert. Damit unkodierte AC3 Audiodaten weitergeleitet werden, muss der Eintrag *AC3PASSTHROUGH=yes* gesetzt werden.

A.3 DVD Laufwerksoptionen

Damit DVD-Videos problemlos abgespielt werden und die Laufwerkgeräusche minimiert werden, können mit dem Linux Tool **hdparm** einige Laufwerksoptionen verändert werden [79]. Durch Aktivieren des 32-Bit und DMA-Modus wird der Prozessor entlastet. Die Laufwerksgeschwindigkeit sollte auf 1-fach gesetzt werden, damit wird die Drehgeschwindigkeit reduziert und somit auch die Laufwerksgeräusche. Diese Drehgeschwindigkeit reicht aus, um eine Video-DVD abzuspielen.

Der Aufruf “`hdparm -c 1 -d 1 -E 1 /dev/dvd`” erzielt die gewünschten Effekte; eventuell muss noch der Devicename angepasst werden.

A.4 Installation der DVB Karte

Folgende Abschnitte erläutern die Installation und Konfiguration der DVB Karte. Sie sind Teil der in [65] beschriebenen Arbeit und werden hier zur Vollständigkeit wieder aufgeführt.

A.4.1 Treiber

Linuxtreiber für die Siemens DVB Karte kann man bei linuxtv.org[1] herunterladen. Nachdem man sie mit `./configure` und `make` compiliert hat, kann man sie mit `make insmod` installieren.

A.4.2 Konfiguration

Die Einstellungen für das verwendete LNB befindet sich in der Datei setup.conf des VDRs. Diese Datei ist schon so konfiguriert, dass ein standard Digital-LNB problemlos angesteuert werden kann. Folgende Werte können bei Bedarf verändert werden:

LnbSLOF Umschaltfrequenz (in MHz) zwischen oberem und unterem Band

LnbFrequLo Oszillatorfrequenz des unteren Bandes

LnbFrequHi Oszillatorfrequenz des oberen Bandes

CurrentChannel Kanalnummer, die beim nächsten Start angesprungen werden soll

Die Konfiguration der einzelnen Sender wird in der Datei channel.conf erledigt. Jede Zeile repräsentiert dabei einen Sender, der durch folgende Parameter spezifiziert wird, wobei die einzelnen Werte durch einen Doppelpunkt getrennt werden:

Sendername: Name, der zur Anzeige für das OSD benutzt wird

Transponderfrequenz

Polarisation: Horizontal (“h”) oder vertikal (“v”).

DiSEqC-id: Diese id selektiert das entsprechende LNB, wenn mehrere LNBs benutzt werden.

Symbolrate

video-pid: ID des Videostroms für den Sender

audio-pid: ID des Audiostroms für den Sender

[1] `http://www.linuxtv.org/dvb/siemens_dvb.xml`

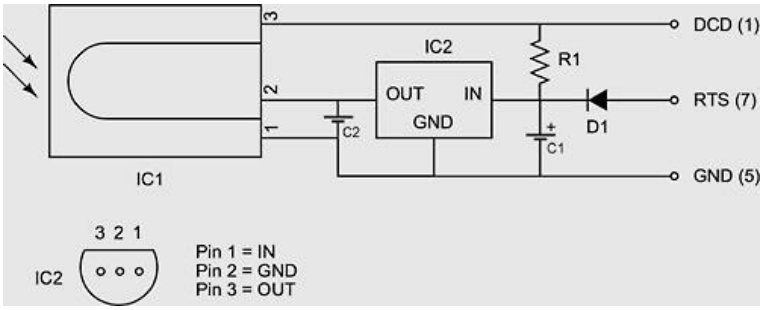

Abbildung A.2: Schaltplan für den IR Empfänger (siehe auch [15]).

Komponente	Spezifikation
Diode (D1)	1N4148
Widerstand (R1)	4,7 kOhm
IR Empfängermodul (IC1)	TSOP1738 oder SFH506-38
Spannungswandler (IC2)	78L05, 100 mA
Kondensator (C1)	10 μF, 16 V
Kondensator (C2)	100 nF

Tabelle A.1: Stückliste des IR Empfängers. (Entnommen aus [65]).

teletext-pid: ID des Bildschirmtextes für den Sender

conditional-access: Id der DVB Karte, die den Sender dekodieren kann (nur bei verschlüsselten Sendern)

pnr: nicht benutzt

Beispiel einer Konfigurationszeile:

```
ARD:11837:h:0:27500:101:102:0:0:28106
```

A.5 Infrarotempfänger

Die nachfolgenden Abschnitte beschreiben den Zusammenbau und die Treiberinstallation eines Infrarotempfängers. Sie sind Teil der in [65] beschriebenen Arbeit und werden hier zur Vollständigkeit wieder aufgeführt.

A.5.1 Schaltplan

Abbildung A.2 zeigt den Schaltplan eines Infrarotempfängers. Alle benötigen Einzelteile sind in Tabelle A.1 aufgeführt. Tabelle A.2 zeigt die Pinbelegung der seriellen Schnittstelle. Weiterführende Informationen zum Thema IR Receiver findet man in [15].

Name	25-Pin	9-Pin	Erklärung
RTS	4	7	Request To Send (Spannung)
GND	7	5	Masse
DCD	8	1	Data Carrier Detect (IR Signal)

Tabelle A.2: Pinbelegung der seriellen Schnittstelle (Entnommen aus [65]).

A.5.2 Treiberinstallation

Der LIRC-Treiber kann von [16] bezogen werden. Nachdem er entpackt wurde, kann er mit `./configure`, `make` und `install` als Root installiert werden. Weitere Informationen zur Treiberinstallation findet man in [65].

Das Einbinden einer Fernbedienung in das LIRC-System kann man in [16] nachlesen. Hier ein Ausschnitt der "lircrc" Konfigurationsdatei, die für die Verbindung von lirc und der MMBox Applikation benötigt wird:

```
begin
  remote = panasonic.lirc
  button = POWER
  prog   = nmmlirc
  repeat = 0
  config = KEY_ESC
end

begin
  remote = panasonic.lirc
  button = POWER_VCR
  prog   = nmmlirc
  repeat = 0
  config = KEY_q
end

...
```

Die Schlüsselwörter *begin* und *end* enthalten die Beschreibung genau einer Taste. Jede Taste der Fernbedienung bekommt einen String zugewiesen, der gesendet wird, wenn die Taste gedrückt wird. Innerhalb der Schlüsselwörter stehen weiter Parameter, die das Verhalten bei einem Tastendruck beschreiben:

remote: Der Name der Fernbedienung, den man bei der Konfiguration des LIRC-Paketes angegeben hat.

button: Fernbedienungstaste, die beschrieben wird.

prog: Name der Applikation, für die die Taste bestimmt ist.

repeat: Steht dieser Wert auf 1, dann wird das Event mehrere Male, solange der Benutzer die Taste drückt, verschickt. Bei einem Wert von 0 wird das Event nur einmal weggeschickt.

config: String, der beim Drücken der Taste verschickt wird.

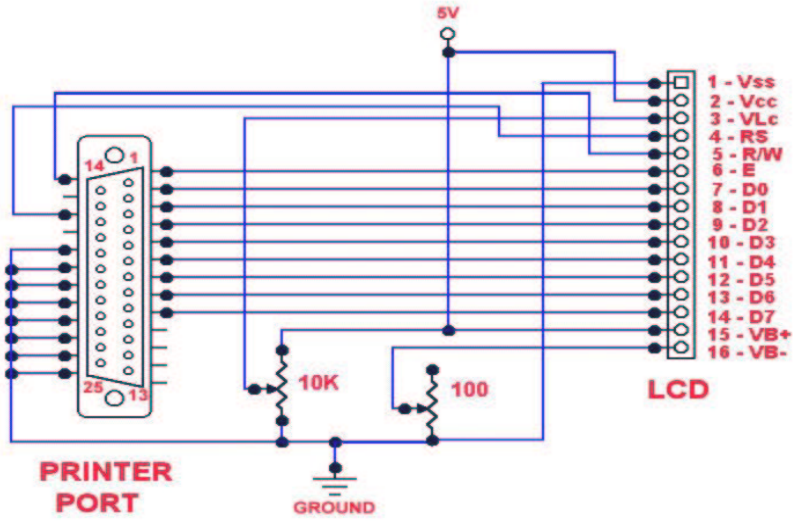

Abbildung A.3: Anschlussplan des LC-Displays an den Parallelport (siehe auch [75])

Komponente	Spezifikation
Potentiometer	100 Ohm
Potentiometer	10 kOhm

Tabelle A.3: Bauteile, die zum Anschließen des LC-Displays an den Parallelport benötigt werden.

A.6 LC-Display

Der Anschluss eines LC-Displays an einen PC und dessen Ansteuerung wurden im Rahmen eines Fortgeschrittenen-Praktikums entwickelt und wird hier zur Vollständigkeit wieder aufgeführt [65].

A.6.1 Hardware

Abbildung A.3 zeigt den Anschlussplan des LC-Displays an den Parallelport. Alle Informationen über den Schaltplan wurden aus [75] genommen.

Das LC-Display ist ein handelsübliches (Reichelt Elektroniks[2]) Zeichendisplay mit 2 Zeilen zu je 40 Zeichen. Angesteuert wird es durch einen KS0076B Chip von Samsung Electronics. Zum Anschluss an den Parallelport sind die in Tabelle A.3 aufgelisteten Bauteile notwendig.

[2] `http://www.reichelt.de/`

A.6.2 Treiber

Der Treiber zum Ansteuern des LC-Dislays wurde im Rahmen des Fortgeschrittenen-Praktikums entwickelt und kann von [8] bezogen werden. Das Headerfile `lcd_2x20.h` liefert eine Struktur **`struct`** `LCDctrl`, die benutzt wird, um Daten und Kommandos an das Display zu schicken:

```
#define LCD_WRITE 10
#define LCD_CLEAR 20
#define LCD_WRITE_CHAR 30
#define LCD_DEC_USE_COUNT 40
#define CHARNUM_UP_ARROW 0x0
#define CHARNUM_DOWN_ARROW 0x1
#define CHARNUM_TRADEMARK_T 0x2
#define CHARNUM_TRADEMARK_M 0x3
#define CHARNUM_BAR1 0x4
#define CHARNUM_BAR2 0x5
#define CHARNUM_BAR3 0x6
#define CHARNUM_BAR4 0x7
#define CHARNUM_BAR5 0xff // can't write more than 8 user
// defined characters , and "all pixels on" was already defined as 0 xff .

const unsigned char LCD_CHAR_BAR1[] = {
  0x10, //---10000
  0x10, //---10000
  0x10, //---10000
  0x10, //---10000
  0x10, //---10000
  0x10, //---10000
  0x10, //---10000
  0x10 //---10000
};

[...]

struct LCDctrl
{
  unsigned char line0[41];
  unsigned char line1[41];
  int posx;
  int posy;
  unsigned char ch;
};
```

Die **`struct`** `LCDctrl` Struktur kann benutzt werden, um ganze Textzeilen anzuzeigen, einzelne Zeichen oder um den Inhalt des Displays zu löschen. In `line0[41]` steht der Inhalt der 1. Zeile in `line1[41]` der Inhalt der 2. Zeile. Angenommen `fd` ist der Filedescriptor für das Device `/dev/lcd/lcd_2x20`, dann kann man diese Zeilen an das Display schicken:

```
ioctl(fd, LCD_WRITE, &lcdctrl);
```

Die Werte `posx`, `posy`, und `ch` werden hier ignoriert. Sie werden benutzt, wenn man einzelne Zeichen an das Display schickt. In `posx` und `posy` steht die Position und in `ch` das Zeichen, das ausgegeben werden soll:

```
ioctl(fd, LCD_WRITE_CHAR, &lcdctrl);
```

Das Display kann mit folgendem Kommando gelöscht werden:

```
ioctl(fd, LCD_CLEAR, &lcdctrl);
```

Ein high-Level Interface zur Ansteuerung des LC-Displays wird in Abschnitt 5.3.3 vorgestellt.

A.7 MMBox Konfiguration

A.7.1 Voreinstellungen

Die MMBox Konfigurationsdatei ".mmboxrc' enthält alle persönlichen Einstellungen, die das Verhalten der MMBox Applikation beeinflusst. Sie wird im Homeverzeichnis unter dem Namen ".mmboxrc" abgelegt oder kann der MMBox Applikation per Kommandozeile mitgegeben werden:

```
# Adresse des CDDB Servers, von dem CD-Player und CD-Grabber ihre
# Titelinformationen erhalten
cddbserver=freedb.freedb.org

# Verzeichnis, in dem die Applikation ihre Multimediafiles sucht und ablegt
rootpath=/home/uder/mmbox

# Verzeichnis relativ zum rootpath, in dem alle aufgenommenen Audiofiles
# plaziert werden
grabdestinationdirectory=Audio-Recordings

# Verzeichnis relativ zum rootpath, in dem alle aufgenommenen Videofiles
# plaziert werden
tvdestinationdirectory=TV-Recordings

# Gibt an, ob die Applikation im Vollbildmodus läuft oder nicht
fullscreen=0

# Verzeichnis, in dem alle Bilder und Zeichensätze liegen, die
# im XML Konfigurationsfile angegeben wurden
resourcepath=/home/user/nmm-0.1.0/resources/mmbox/

# Pfad zur XML Konfigurationsdatei
configfile=/home/user/nmm-0.1.0/resources/mmbox/configuration.xml

# (De-)aktivieren der LIRC Infrarotunterstützung
uselirc=0

# Benutze spezielles Matroxdevice anstatt des XDipslays
matrox=0
```

A.7.2 XML Beschreibung der Zustände und Tastenzuordnung

```
<?xml version="1.0"?>
<configuration>
  <resourcepath>/usr/local/share/nmm/mmbox/</resourcepath>
  <keyboard>
    <event key="a">KEY_a</event>
    <event key="b">KEY_b</event>
    <event key="c">KEY_c</event>
    <event key="d">KEY_d</event>
    <event key="e">KEY_e</event>
    <event key="f">KEY_FF</event>
    <event key="g">KEY_g</event>
    <event key="h">KEY_h</event>
    <event key="i">KEY_i</event>
    <event key="j">KEY_j</event>
    <event key="k">KEY_k</event>
    <event key="l">KEY_Stop</event>
    <event key="m">KEY_m</event>
    <event key="n">KEY_n</event>
    <event key="o">KEY_OSD</event>
    <event key="p">KEY_p</event>
    <event key="q">KEY_q</event>
    <event key="r">KEY_REW</event>
    <event key="s">KEY_s</event>
    <event key="t">KEY_t</event>
    <event key="u">KEY_u</event>
    <event key="v">KEY_v</event>
    <event key="w">KEY_w</event>
    <event key="x">KEY_x</event>
    <event key="y">KEY_y</event>
    <event key="z">KEY_z</event>

    <event key="␣">KEY_space</event>

    <event key="0">KEY_0</event>
    <event key="1">KEY_1</event>
    <event key="2">KEY_2</event>
    <event key="3">KEY_3</event>
    <event key="4">KEY_4</event>
    <event key="5">KEY_5</event>
    <event key="6">KEY_6</event>
    <event key="7">KEY_7</event>
    <event key="8">KEY_8</event>
    <event key="9">KEY_9</event>

    <event key="F1">KEY_F1</event>
    <event key="F2">KEY_F2</event>
    <event key="F3">KEY_F3</event>
```

```
  <event key="F4">KEY_F4</event>
  <event key="F5">KEY_F5</event>
  <event key="F6">KEY_F6</event>
  <event key="F7">KEY_F7</event>
  <event key="F8">KEY_F8</event>
  <event key="F9">KEY_F9</event>
  <event key="F10">KEY_F10</event>
  <event key="F11">KEY_F11</event>
  <event key="F12">KEY_F12</event>

  <event key="LEFT">KEY_Left</event>
  <event key="RIGHT">KEY_Right</event>
  <event key="UP">KEY_Up</event>
  <event key="DOWN">KEY_Down</event>
  <event key="KEY_VCR_Up">KEY_VCR_Up</event>
  <event key="KEY_VCR_Down">KEY_VCR_Down</event>

  <event key="ESC">KEY_ESC</event>
  <event key="RETURN">KEY_Return</event>
</keyboard>

 <menu id="MainMenu" columns="2" background="neon3/back/
    campfires2k2.png" needvideo="no" needaudio="no">
  <entry index = "1"
       on   = "neon3/tv.on.png"
       off  = "neon3/tv.off.png"
       x    = "160"
       y    = "58">DVBMenu</entry>

  <entry index = "2"
       on   = "neon3/playlist.on.png"
       off  = "neon3/playlist.off.png"
       x    = "350"
       y    = "58">Playlist</entry>

  <entry index = "3"
       on   = "neon3/dvd.on.png"
       off  = "neon3/dvd.off.png"
       x    = "98"
       y    = "156">DVDMenu</entry>

  <entry index = "4"
       on   = "neon3/tasklist.on.png"
       off  = "neon3/tasklist.off.png"
       x    = "350"
       y    = "156">TaskManager</entry>

  <entry index = "5"
       on   = "neon3/cd.on.png"
```

```
        off  = "neon3/cd.off.png"
        x    = "154"
        y    = "260">AudioCDMenu</entry>

  <entry index = "6"
        on   = "neon3/config.on.png"
        off  = "neon3/config.off.png"
        x    = "350"
        y    = "260">Configuration</entry>

  <entry index = "7"
        on   = "neon3/mp3.on.png"
        off  = "neon3/mp3.off.png"
        x    = "84"
        y    = "354">MP3Player</entry>
</menu>

<menu id="DVDMenu" background="neon3/back/cobaltdaisy.png"
     needvideo="no" needaudio="no">
  <entry index = "1"
        on   = "neon3/play.on.png"
        off  = "neon3/play.off.png"
        x    = "60"
        y    = "200">DVDPlayer</entry>

</menu>

<menu id="DVBMenu" background="neon3/back/cobaltdaisy.png"
     needvideo="no" needaudio="no">
  <entry index = "1"
        on   = "neon3/live.on.png"
        off  = "neon3/live.off.png"
        x    = "60"
        y    = "200">DVB-TV</entry>

  <entry index = "2"
        on   = "neon3/timer.on.png"
        off  = "neon3/timer.off.png"
        x    = "330"
        y    = "200">TV-Timer</entry>
</menu>

<menu id="AudioCDMenu" background="neon3/back/cobaltdaisy.
    png" needvideo="no" needaudio="no">
  <entry index = "1"
        on   = "neon3/play.on.png"
        off  = "neon3/play.off.png"
        x    = "60"
        y    = "200">CDPlayer</entry>
```

```
  <entry index = "2"
        on   = "neon3/grab.on.png"
        off  = "neon3/grab.off.png"
        x    = "330"
        y    = "200">CDGrabber</entry>
</menu>


<cdplay id="CDPlayer" background="neon3/back/cobaltdaisy.
    png" needvideo="no" needaudio="yes">
</cdplay>

<cdgrab id="CDGrabber" background="neon3/back/cobaltdaisy.
    png" needvideo="no" needaudio="no">
</cdgrab>

<mp3play id="MP3Player" background="neon3/back/cobaltdaisy
    .png" needvideo="no" needaudio="yes">
</mp3play>

<dvdplay id="DVDPlayer" background="neon3/back/cobaltdaisy
    .png" needvideo="yes" needaudio="yes">
</dvdplay>

<dvbplay id="DVB-TV" background="neon3/back/cobaltdaisy.
    png" needvideo="yes" needaudio="yes">
</dvbplay>

<tvtimer id="TV-Timer" background="neon3/back/cobaltdaisy.
    png" columns="2" needvideo="no" needaudio="no">
</tvtimer>

<dvbrecord id="DVB-Recorder" needvideo="no" needaudio="no"
    >
</dvbrecord>

<taskmgr id="TaskManager" background="neon3/back/
    cobaltdaisy.png" needvideo="no" needaudio="no">
</taskmgr>

<config id="Configuration" background="neon3/back/
    cobaltdaisy.png" needvideo="no" needaudio="no">
  <option id = "AC3 Passthrough"
         short = "useac3">no yes</option>
  <option id = "Use CDDB"
         short = "cddb">yes no</option>
  <option id = "MP3 Enc. Bitrate"
         short = "grabmp3bitrate">128 192 64</option>
```

```
  <option id = "Use hardware MPEG decoder"
        short = "hard_mpeg">no yes</option>
</config>

<playlist id="Playlist" background="neon3/back/cobaltdaisy
    .png" needvideo="yes" needaudio="yes">
  <win id  = "Selection"
      x    = "50"
      y    = "320"
      width = "100"
      height = "200"

      win_active_color = "000 000 255 128"
      win_inactive_color = "128 128 128 128"

      sel_active_color = "128 255 000 255"
      sel_active_color2 = "255 128 000 255"
      sel_inactive_color = "128 128 128 255"></win>

  <win id  = "Track"
      x    = "170"
      y    = "320"
      width = "500"
      height = "200"

      win_active_color = "000 000 255 128"
      win_inactive_color = "128 128 128 128"

      sel_active_color = "128 255 000 255"
      sel_active_color2 = "255 128 000 255"
      sel_inactive_color = "128 128 128 255"></win>

  <win id  = "Playlist"
      x    = "50"
      y    = "50"
      width = "620"
      height = "250"

      win_active_color = "000 000 255 128"
      win_inactive_color = "128 128 128 128"

      sel_active_color = "128 255 000 255"
      sel_active_color2 = "255 128 000 255"
      sel_inactive_color = "128 128 128 255"></win>
</playlist>

<osd id="OSDInputState" needvideo="no" needaudio="no">
  <button id = "OSD_OK"
        bitmap = "button_ok.png"></button>
```

```
    <button id = "OSD_CANCEL"
          bitmap = "button_cancel.png"></button>
  </osd>

  <osdsymbols>
    <symbol id="PLAY">play.png</symbol>
    <symbol id="PAUSE">pause.png</symbol>
    <symbol id="FF">forward.png</symbol>
    <symbol id="REW">backward.png</symbol>
  </osdsymbols>

</configuration>
```

A.8 Tastenbelegung

Die Standardbelegung für PC-Tastatur zum Steuern der MMBox Applikation sieht wie folgt aus:

Cursor Tasten: Navigationstasten (Rechts, Links, Hoch, Runter)

Enter Taste: Play

L-Taste: Stop

P-Taste: Pause

F-Taste: FF (meist zum Vorwärtsspulen benutzt)

R-Taste: REW (meist zum Rückwärtsspulen benutzt)

Q-Taste: Back

A-Taste: Funktion 1

S-Taste: Funktion 2

O-Taste: OSD

ESC-Taste: Exit

F12: Aus

A.9 AC3 Frame Codes

Code	Bitrate (kbps)	Worte bei 48 KHz	Worte bei 44,1 KHz	Worte bei 32 KHz
000000	32	64	69	96
000001	32	64	70	96
000010	40	80	87	120
000011	40	80	88	120
000100	48	96	104	144
000101	48	96	105	144
000110	56	112	121	168
000111	56	112	122	168
001000	64	128	139	192
001001	64	128	140	192
001010	80	160	174	240
001011	80	160	175	240
001100	96	192	208	288
001101	96	192	209	288
001110	112	224	243	336
001111	112	224	244	336
010000	128	256	278	384
010001	128	256	279	384
010010	160	320	348	480
010011	160	320	349	480
010100	192	384	417	576
010101	192	384	418	576
010110	224	448	487	672
010111	224	448	488	672
011000	256	512	557	768
011001	256	512	558	768
011010	320	640	696	960
011011	320	640	697	960
011100	384	768	835	1152
011101	384	768	836	1152
011110	448	896	975	1344
011111	448	896	976	1344
100000	512	1024	1114	1536
100001	512	1024	1115	1536
100010	576	1152	1253	1728
100011	576	1152	1254	1728
100100	640	1280	1393	1920
100101	640	1280	1394	1920

Tabelle A.4: Codetabelle für Bitrate und Größe eines AC3 Frames. (Quelle: [7])

Abbildungsverzeichnis

1.1 Die Multimedia-Box 3
1.2 Menü Styles 4

2.1 Das Benutzerinterface des Linux MPlayers 7
2.2 Benutzerinterface des Linux Alsaplayers 7
2.3 Benutzerinterface des Video Disk Recorders 9

3.1 Ein detaillierter NMM Knoten. 17
3.2 Ein einfacher MP3 Player 18
3.3 Ein NMM-Flussgraph 20
3.4 Composite Knoten 22
3.5 Zustände und Zustandsübergänge von NMM Nodes. 23

4.1 Multimedia-Box mit LC Display 32
4.2 Systemaufbau 33
4.3 Die digitale Satellitenkarte (DVB-s). 36
4.4 Das MPEG-Encoderboard mit KFir Chipsatz. 37
4.5 LC Display 38
4.6 Das Innenleben der Multimedia-Box. 38

5.1 Transparente grafische Objekte 41
5.2 Schema des OSDManagerNode 42
5.3 Ein Window mit Widget 43
5.4 Windowupdate 45
5.5 Aufbau des YV12 Farbformats. 46
5.6 Klassenhierarchie von `Window` 47
5.7 Widget-Interface 49
5.8 Weiterleiten der Anfrage an ein Widget 52
5.9 Composite Widget 52
5.10 3D Rechteck des Bitmapreaders 53
5.11 Zeichensatzbitmap 54
5.12 Interface der BitmapFont Klasse. 56
5.13 Buttons Widget 57
5.14 ProgressBar Widget 59

5.15 TextView Widget 60
5.16 TextView Widget mit geänderter Linespace 60
5.17 TextView Widget mit Selectionbar 61
5.18 TimeView Widget 62
5.19 Zusammenstellung verschiedener Widgets 63
5.20 Ein TextWidget mit Plane- und BorderDecorator. 64
5.21 Ein TextWidget mit Plane-, Border-, Scroll- und HeaderDecorator. 65
5.22 MessageUnit 66
5.23 DoubleListUnit 67
5.24 OSDManagerNode mit einem Eingang 74
5.25 OSDManagerNode mit zwei Eingängen 75
5.26 Windows im OSDManagerNode 75
5.27 PUNPCKLBW MMX Befehl 77
5.28 PUNPCKHWD MMX Befehl 78
5.29 PUNPCKHDQ MMX Befehl 78
5.30 PACKUSWB MMX Befehl 79
5.31 Run Length Encoding 83
5.32 Farbzuweisung der DVD Untertitel 84
5.33 Farbzuweisung der DVD Menüknöpfe 85
5.34 Aufbau eines SPU Pakets 86
5.35 DVD mit Menühighlight 87
5.36 Ausgänge des MPEG Demuxers 89
5.37 Ausgänge des MPEG Demuxers im DVD Modus 90
5.38 Zustände des MPEGTimeshiftingNode 91
5.39 Aufbau eines AC3 Frames 94
5.40 Aufbau des AC3 Headers 94
5.41 Display ohne Shared Memory 95
5.42 Display ohne Shared Memory 96

6.1 Hauptmenü 103
6.2 Kapitelmenü einer Video DVD 104
6.3 CD Player 105
6.4 TV-Viewer 106
6.5 TV-Timer 107
6.6 Playlist 109
6.7 MP3-Spieler 110
6.8 Konfiguration 111
6.9 MMBoxState Klasse 114
6.10 Klassendiagramm der MMBox-Zustände 115
6.11 Standardgraph 116
6.12 XML Parser 117
6.13 Architektur der MMBox-Applikation 118
6.14 MMBox-Applikation mit mehreren aktiven Zuständen 119
6.15 Vom Eingabegerät zum Methodenaufruf 120

6.16 State Umschalten 121
6.17 Skins Neon2, Neon3 122
6.18 Skins Icon, Timber 124
6.19 NMM Graph eines DVD Players 125
6.20 NMM Graph eines CD Players 127
6.21 NMM Graph eines CD Grabbers 128
6.22 NMM Graph des TV Viewers 128
6.23 NMM Graph des TV Viewers mit Hardwarebeschleunigung . . . 129
6.24 NMM Graph des MP3 Players 130
6.25 NMM Graph für Playlist ohne Video 131
6.26 NMM Graph für Playlist mit Audio und Video 132
6.27 Timerliste und Stateübergang 135
6.28 Hinzufügen von Timerevents 136
6.29 Speichern eines TV-MPEG-Stroms 137
6.30 Flussgraph einer Slideshow 141

A.1 NVidia TV Tool 152
A.2 Schaltplan des IR Empfängers 154
A.3 Schaltplan des LC-Displays 156

Literaturverzeichnis

[1] *DivX Video Codec*. http://www.divx.com.

[2] *STOREit project*. http://www.extra.research.phillips.com/euprojects/mytv/.

[3] *TV Anytime Forum*. http://www.tv-anytime.org/.

[4] *XMLPP*, 2002. http://sourceforge.net/projects/xmlpp/.

[5] AARON HOLTZMAN und MICHEL LESPINASSE ET AL: *liba52 - A free ATSC A/52 stream decoder*, 2002. http://liba52.sourceforge.net/.

[6] AARON HOLTZMAN und MICHEL LESPINASSE ET AL: *libmpeg2 - A free MPEG-2 video stream decoder*, 2002. http://libmpeg2.sourceforge.net/.

[7] ADVANCED TELEVISION SYSTEMS COMMITTEE: *Digital Audio Compression (AC-3)*. http://www.atsc.org/.

[8] ANDREAS POMI und MARKUS SAND: *Linux LCD Driver*, 2002. http://graphics.cs.uni-sb.de/NMM/lcd/.

[9] ANDREAS ROEDL: *The Linux Home-Entertainment-Server*, 2002. http://www.flood-net.de/index.html?/flood/projects/liquid/.

[10] ANDREAS S. GLASSNER: *Graphics Gems*. Academic Press, Erste Auflage, 1990.

[11] BENJAMIN DEUTSCH: *An Introduction to the Network-Multimedia Software Architecture*, 2002. Fortgeschrittenenpraktikum.

[12] BERTRAM WOHAK und REINHOLD MAURUS: *80x86/Pentium Assembler*. IWT Thomson Publishing, Erste Auflage, 1995.

[13] BRADFORD NICHOLS: *Pthreads Programming. A POSIX Standard for Better Multiprocessing*. O'Reilly and Associates, 1996.

[14] *CDDA Paranoia*, 2002. http://www.xiph.org/paranoia/.

[15] CHRISTOPH BARTELMUS: *Ein Pinguin sieht (infra)rot - PC als fernbedienbare Hi-Fi-Anlage.* c't - Magazin für Computertechnik, 18:208ff, 2000.

[16] CHRISTOPH BARTELMUS ET AL.: *Linux Infrared Remote Control*, 4 2002. http://www.lirc.org/.

[17] DANIEL BERTRAND und RUI SOUSA: *Emu10k1 sound driver*. http://sourceforge.net/projects/emu10k1.

[18] *Digital Blasphemy*, 2002. http://www.digitalblasphemy.com/.

[19] DIRK THIERBACH, CHRISTER PALM, HARRI SALOKORPI und OLOV LINBERG: *NV-TV*. http://sourceforge.net/projects/nv-tv-out/.

[20] ENERMAX: *PC Netzteillüfter*. http://www.enermax.com.tw/.

[21] E. PERSOON: *Conference Paper: on how storage at home will change how we watch television.*

[22] ERICH GAMMA, RICHARD HELM, RALPH JOHNSON und JOHN VLISSIDES: *Design Patterns.* Addision-Wesley, 21. Auflage, 2000.

[23] FOCUS ENHANCEMENT: *Homepage*, 2002. http://www.focusinfo.com/.

[24] *freedb.org*, 2002. http://www.freedb.org/.

[25] *Freevo*, 2002. http://sourceforge.net/projects/freevo/.

[26] FUJITSU-SIEMENS: *Activy 300*, 2001. http://www.fujitsu-siemens.com/rl/products/broadband/function300.html.

[27] GERHARD STOLL: *SAMBITS; An advanced broadcasting system which provides a perfect TV and Web experience for the viewer*, 2001.

[28] GIGABYTE: *Homepage*, 2002. http://tw.giga-byte.com/.

[29] *GStreamer*, 2002. http://www.gstreamer.net/apps/.

[30] GUY ERIC SCHALNAT und GLENN RANDERS-PEHRSON ET AL: *libpng - Portable Network Graphics (PNG) Reference Library*, 2002. http://www.libpng.org/.

[31] HAUPPAUGE: *Homepage*, 2002. http://www.hauppauge.de.

[32] H. SCHULZRINNE, STEPHEN L. CASNER, RON FREDERICK und VAN JACOBSON: *RFC 1889 - RTP: A Transport Protocol for Real-Time Applications*, 1996.

[33] H. SCHULZRINNE, A. RAO und R. LANPHIER: *RFC 2326 - Real Time Streaming Protocol (RTSP)*, 1998.

[34] IBKS: *PC Gehäuse*. http://www.ibks.de/.

[35] *IEC 958 (Digital Audio Interface)*.

[36] INSTITUT FÜR RUNDFUNKTECHNIK: *Homepage*. http://www.irt.de/.

[37] INTEL: *Intel Architecture, Software Developer's Manual, Instruction Set Reference*, 1997.

[38] INTEL: *MMX technology, Programmers Reference Manual*, 1997. http://www.intel.com.

[39] INTEL: *Media/Entertainment Gateway*, 2001.

[40] INTEL: *Homepage*, 2002. http://www.intel.com.

[41] JAMES D. FOLEY, ANDRIES VAN DAM, STEVEN K. FEINER und JOHN F. HUGHES: *Computer Graphics: Principles and Prtactice*. Addison-Wesley, Zweite Auflage, 1997.

[42] JIM TAYLOR: *DVD Demystified*. Web Enhanced, Erste Auflage, 1998.

[43] JONATHAN CORBET: *The MIT Shared Memory Extension*.

[44] KEITH JACK: *Video Demystified*. LLH Technology Publishing, Zweite Auflage, 1996.

[45] KIEN A. HUA, YING CAI und SIMON SHEU: *A Multicast Technique for True Video-on-Demand Services*, 1998.

[46] KLAUS SCHMIDINGER: *Video Disk Recorder*. http://www.cadsoft.de/people/kls/vdr/.

[47] LG ELECTRONICS: *Homepage*. http://www.lge.de/.

[48] *libdvdnav*, 2002. http://dvd.sourceforge.net/.

[49] *linuxtv.org*, 2002. http://www.linuxtv.org/.

[50] MACROMEDIA: *Macromedia Flash*. http://www.macromedia.com/.

[51] MARCO LOHSE, MICHAEL REPPLINGER und PHILIPP SLUSALLEK: *An Open Middleware Architecture for Network-Integrated Multimedia. Protocols and Systems for Interactive Distributed Multimedia Systems (IDMS/PROMS)*, 2002.

[52] MARCO LOHSE und PHILIPP SLUSALLEK: *An Open Platform for Multimedia Entertainment Systems. EUROPRIX Conference*, 2002.

[53] MARCO LOHSE, PHILIPP SLUSALLEK und PATRICK WAMBACH: *Extended Format Definition and Quality-driven Format Negotiation in Multimedia Systems*. *Multimedia 2001 – Proceedings of the Eurographics Workshop*, 2001.

[54] *Multimedia Home Platform*, 2002. `http://www.mhp.org`.

[55] MICHAEL B. JONES: *The Microsoft Interactive TV System*, 1997.

[56] MICHAEL N. NELSON, MARK A. LINTON und SUSAN S. OWICKI: *A Highly Available, Scalable ITV System*, 1995.

[57] MICHAEL REPPLINGER: *Arbeitstitel: Architektur zur Anbindung und Kontrolle im Netz verteilter Multimedia-Geräte*. Diplomarbeit, Universität des Saarlandes, 2003.

[58] MICHAEL ZINK, CARSTEN GRIWODZ und RALF STEINMETZ: *KOM Player - A Platform for Experimental VoD Research*, 2001.

[59] MICROSOFT: *Windows XP Media Center*. `http://www.microsoft.com/windowsxp/mediacenter/`.

[60] MICROSOFT: *Windows Mediaplayer*, 2002. `http://www.microsoft.com`.

[61] MIKE CHENG, ROBERT HEGEMANN, FRANK KLEMM, ALEXANDER LEIDINGER, NAOKI SHIBATA, MARK TAYLOR und TAKEHIRO TOMINIGA: *The Lame Project*, 2002. `http://www.mp3dev.org/mp3/`.

[62] *Network-Integrated Multimedia Middleware*, 2002. `http://www.networkmultimedia.org/`.

[63] NULLSOFT: *Winamp*, 2002. `http://www.winamp.com`.

[64] NVIDIA: *Homepage*, 2002. `http://www`.

[65] PATRICK BECKER, PATRICK CERNKO, WOLFGANG ENDERLEIN, MARC KLEIN und MARKUS SAND: *Design and Development of a Multimedia Home Entertainment System for Linux*, 2002. Fortgeschrittenenpraktikum, Universität des Saarlandes.

[66] PATRICK CERNKO: *Arbeitstitel: Implementation of a Distributed Multimedia Home-Entertainment System*. Diplomarbeit, Universität des Saarlandes, 2003.

[67] PATRICK G.T. HEALEY, ANDREEA VADUVA und YAKUP PAKER: *User Interface Design for CustomTV*, 1995.

[68] PATRICK WAMBACH: *Formatdefinition und Formatverhandlung von Multimedia-Geräten*. Diplomarbeit, Universität des Saarlandes, 2001.

[69] RALF STEINMETZ: *Multimedia Technologie*. Springer, Zweite Auflage, 1999.

[70] ROBERT LESLIE: *MAD: MPEG Audio Decoder*, 2002. http://www.mars.org/home/rob/proj/mpeg/.

[71] RONALD M. TOL und EDWIN A. MONTIE: *TV Anytime: Store it on my TV*.

[72] ÁRPÁD GEREÖFFY ET AL: *MPlayer*, 2002. http://www.MPlayerHQ.hu/homepage/.

[73] SIGMA DESIGNS: *Homepage*, 2002. http://www.sigmadesigns.com/.

[74] STEPHAN DIDAS: *Synchronization in the Network-Integrated Multimedia Middleware (NMM)*, 2002. Fortgeschrittenenpraktikum, Universität des Saarlandes.

[75] SVEN SCHÄFER: *Bau eines parallel angeschlossenen externen LCD Displays*, 2002. http://www.easy-mod.de/default.php?seite=lcd1.

[76] T. BOUTELL ET. AL: *PNG (Portable Network Graphics) Specification. Version 1.0*. Network Working Group, 1997. ftp://ftp.isi.edu/in-notes/rfc2083.txt.

[77] TI KAN: *CDDB Server Protocol*. Bericht.

[78] TIM SCHWARTZ: *Analyse von Audio-Dekompressions-Verfahren und deren Implementierung auf einem Spezialprozessor*. Diplomarbeit, Universität des Saarlandes, 2002.

[79] TOMI LEPPIKANGAS und MARK LORD: *hdparm - get/set hard disk parameters*. Benutzerhandbuch.

[80] TROLLTECH: *QT*, 2002. http://www.trolltech.com.

[81] TUXIA: *Tuxia*, 2002. http://www.tuxia.com/.

[82] VERAX: *Homepage*. http://www.verax.de/.

[83] W3C: *Scalable Vector Graphics (SVG) 1.0 Specification*, 2001. http://www.w3.org/TR/SVG/.

[84] *The XFree Project, Inc.*, 2002. http://www.xfree86.org/.

[85] XUNKER: *The M3U file format*. http://hanna.pyxidis.org/tech/m3u.html.